ANALOG SIGNAL PROCESSING ON A RECONFIGURABLE PLATFORM

A Thesis
Presented to
The Academic Faculty

By

Craig R. Schlottmann

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2009

ANALOG SIGNAL PROCESSING ON A RECONFIGURABLE PLATFORM

Approved by:

Dr. Paul E. Hasler, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Maysam Ghovanloo
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: July 2009

ACKNOWLEDGMENTS

I would first like to thank my advisor, Paul Hasler, for his support and guidance. The rest of my committee, David Anderson and Maysam Ghovanloo, also deserve thanks for their input and comments. Of course, this work would be impossible without my ICE Lab colleagues. I thank them for creating a fun and interesting atmosphere. I can never thank my family (Mom, Dad, and Dawn) enough, who without their constant support and encouragement, it would have been impossible to get this far. Most of all, I am eternally indebted to Shannon, for always standing by me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Cooperative Analog/Digital Signal Processing (CADSP) research group's approach to signal processing is to see what opportunities lie in adjusting the line between what is traditionally computed in digital and what can be done in analog. By allowing more computation to be done in analog, we can take advantage of its low power, continuous domain operation, and parallel capabilities. One setback keeping Analog Signal Processing (ASP) from achieving more wide-spread use, however, is its lack of programmability. The design cycle for a typical analog system often involves several iterations of the fabrication step, which is labor intensive, time consuming, and expensive. These costs in both time and money reduce the likelihood that engineers will consider an analog solution. With CADSP's development of a reconfigurable analog platform, a Field-Programmable Analog Array (FPAA), it has become much more practical for systems to incorporate processing in the analog domain.

In this Thesis, I present an entire chain of tools that allow one to design simply at the system block level and then compile that design onto analog hardware. This tool chain uses the Simulink design environment and a custom library of blocks to create analog systems. I also present several of these ASP blocks covering a broad range of functions from matrix computation to interfacing. In addition to these tools and blocks, the most recent FPAA architectures are discussed. These include the latest RASP general-purpose FPAAs as well as an adapted version geared toward high-speed applications.

CHAPTER 1

INTRODUCTION

In our increasingly digital world, it is easy to think of analog computation as a step in the wrong direction. However, quite the opposite is true; with today's ever-present emphasis on mobility and low power, analog is getting a second look. By utilizing analog's distinct advantages, such as its ability to compute in parallel, its low-current operation, and its ease of scaling and summing, we can perform certain functions in a much more efficient manor than on a digital processor. Figure 1 shows Gene's law, an illustration of the rate of advancement that we can expect from DSPs in terms of power consumption per Million of Multiply Accumulate Cycles a Second (MMACS) [1]. Added to this plot is data showing the power consumed by an analog implementation of the same computation. The analog computation's power need is four orders of magnitude less than its digital counterpart, resulting in a 20-year leap in efficiency.

The Cooperative Analog/Digital Signal Processing (CADSP) research group has been dedicated to harnessing the power of analog, in conjunction with digital, to obtain an overall higher level of performance for signal processing. Our approach is to redefine the line between what is traditionally processed in digital and what can be done with Analog Signal Processing (ASP). In modern signal processing, continuous signals from the outside world are often quantized immediately and given directly to the DSP, leaving analog to handle the conversion. By applying this method, the ADC is usually required to sample at a very high speed and bit precision in order to capture the wide band of the incoming signals, and the DSP is left to process a large amount of data. As an alternate approach, we propose utilizing analog's continuous, low-power processing as a front end to the quantization and subsequent digital processing. With this approach, we are not only taking advantage of ASP's computational strength, but we can also relax the requirements on the converter to a smaller baseband since its input will be more uniform. In this Thesis, I describe an

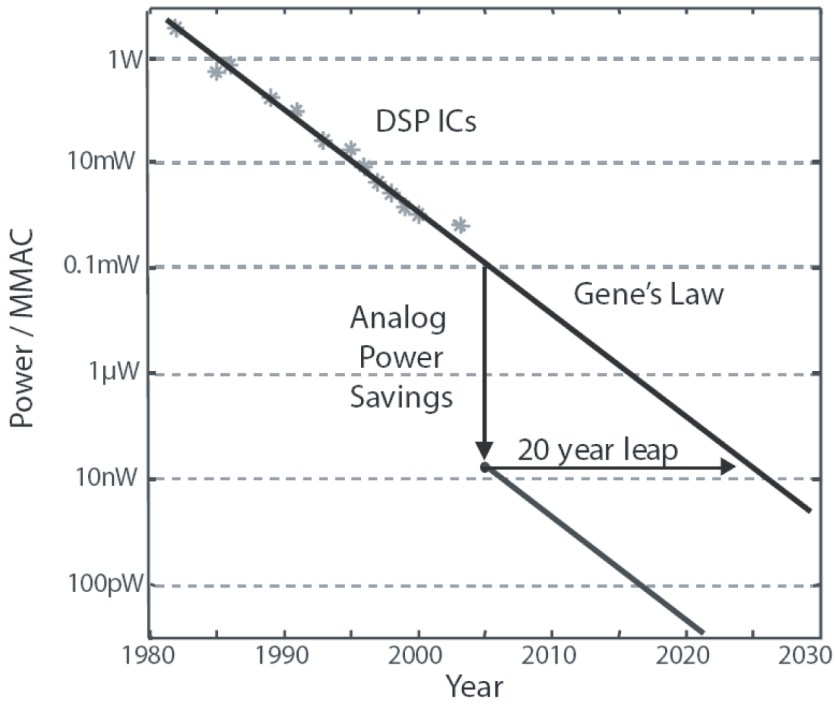


Figure 1. Gene's law. Shows the power trend for digital circuits. Analog poses a 20-year leap in performance.

entire system which enables signal processing tasks to be easily converted into the analog domain by utilizing the reconfigurable analog architecture and design tool chain developed by CADSP.

In Chapter 2, I discuss our fundamental piece of technology, the floating-gate transistor and its device characteristics. I review a programming scheme of tunneling and injecting used to accurately control the charge on these floating gates. Further, I illustrate how we can target a specific transistor for injection by arranging the devices in 2-D arrays.

In Chapter 3, I discuss the current state of Field-Programmable Analog Arrays (FPAAs). I outline the main advantages that FPAAs have to offer the community and how floating gates play an important role in not only the interconnects but in programming biases. I give a detailed description of the Reconfigurable Analog Signal Processor (RASP) FPAA and three of the more recent versions: the 2.8a, 2.9a, and 2.9b.

Chapter 4 details several important analog building blocks and their FPAA implementation. These building blocks form a base for analog signal processing on the FPAA. The Chapter begins by describing two computational blocks, the winner-take-all and the capacitative summer. I then present two interfacing blocks, the V-to-I and I-to-V, which are used to create and read on-chip currents with the on-board voltage tools. I also give simple implementations for voltage and current sources. I provide a detailed discussion of the vector-matrix multiplier (VMM) as it is implemented on the FPAA. Lastly, I demonstrate the use of MITEs both on the general-purpose FPAA and a special MITE FPAA.

Chapter 5 introduces the tools and complete design flow for implementing large systems on the FPAA. I start from the top level and present *sim2spice*, a compiler and library that allows one to create analog systems in the Simulink environment. The compiler then turns this design into a Spice netlist, which I show can be targeted to the FPAA with the RASPER tool. I also describe the evaluation board, which has been built to make the whole FPAA system self-contained. I wrap up this Chapter with two examples of ASP systems that have been compiled down through the whole tool chain.

Finally, Chapter 6 recaps the overall impact of this Thesis. I make note of my personal contributions and provide some thought on what work is still left to be done regrading the topics presented here.

CHAPTER 2

FLOATING-GATE ELEMENTS

The fundamental piece of technology for our reconfigurable system is the floating-gate transistor. Originally reported in 1967 [2], these are transistors whose gates are entirely surrounded by electrical insulator with no DC path to ground, which allows stored charge to be retained. Floating gates have firmly established themselves in digital circuits as a reliable non-volatile memory storage, used in flash and EEPROM. Recent research has been exploring their applications in analog circuits such as multiplier weights, neuromorphic synapses, analog memory, bias generation, and offset removal.

An important feature of these devices is that they can be fabricated in a standard CMOS process. The layout for a floating gate is illustrated in Figure 2. The red poly 1 is the gate of this pFET. It has no direct contacts and is completely surrounded by oxide. This floating node is shown to have two voltage signals capacitively coupled onto it, a tunneling voltage and an input voltage. A MOS capacitor is used for tunneling due to its high quality oxide and a poly capacitor is used for the input voltage.

2.1 Floating-gate Transistor Characteristics

In order to effectively use floating-gate transistors, the I-V relationship along with the stored charge's affects needs to be defined. For the floating-gate pFET in Figure 3, the subthreshold drain current (I_D) is given as

$$I_D = I_o e^{\frac{V_S - \kappa V_{fg}}{U_T}} e^{\frac{V_D}{V_A}} \tag{1}$$

where κ is the capacitive division between the oxide capacitance and the depletion capacitance ($\frac{C_{ox}}{C_{ox}+C_{dep}}$), $U_T = kT/q$ is the thermal voltage, V_A is the Early voltage, and V_{fg} is the voltage at the floating gate given by

$$V_{fg} = \frac{1}{C_T}\left(C_C V_C + C_{tun} V_{tun} + Q\right) \tag{2}$$

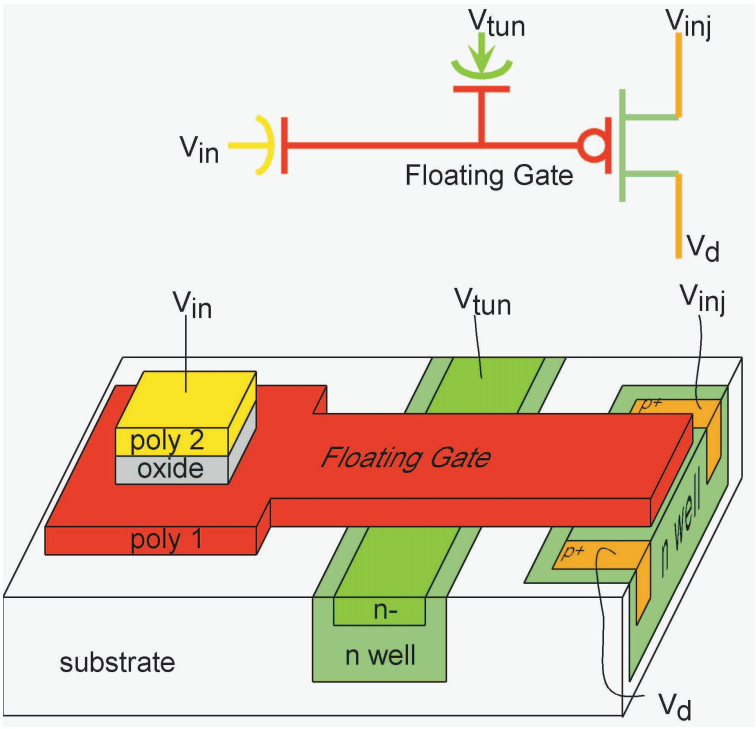


Figure 2. Floating-gate transistor layout. (a) Schematic of a floating-gate transistor. (b) The corresponding layout. Poly 1 is completely insulated by oxide, with no contacts. The input voltage is coupled in through a poly capacitor and the tunneling voltage through a MOS capacitor.

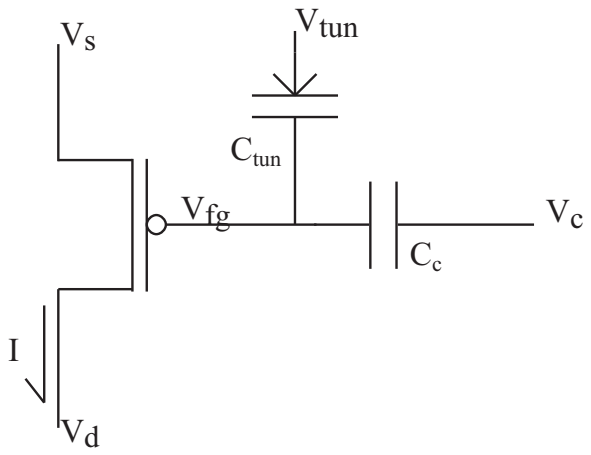


Figure 3. Floating gate schematic.

where $C_T = C_C + C_{tun}$ is the total capacitance at the gate and Q is the charge stored on the floating node. Note that all of the voltages are referenced to the bulk, which in the case of this pFET is V_{DD}. For the situation where V_{tun} is $0V$, Equation 2 can be written as

$$V_{fg} = \frac{C_C}{C_T} V_C + V_{offset} \tag{3}$$

where V_{offset} is Q/C_T.

Equation 3 shows that this floating-gate transistor works very similarly to a traditional pFET, but with a programmable offset on the gate. Precise control over this offset (the stored charge) is what makes floating gate technology so important as an analog memory.

2.2 Floating Gate Charge Modification

There are two well-documented procedures that we use to modify the charge on a floating-gate transistor. Fowler-Nordheim tunneling is used for the removal of electrons and hot-electron injection is used for the addition. With this combination of processes we can effectively program a transistor to have any stored voltage at its gate.

Tunneling is the process used for removing charge from the floating gate. This occurs when an electron is made to pass through a barrier rather than following its conduction

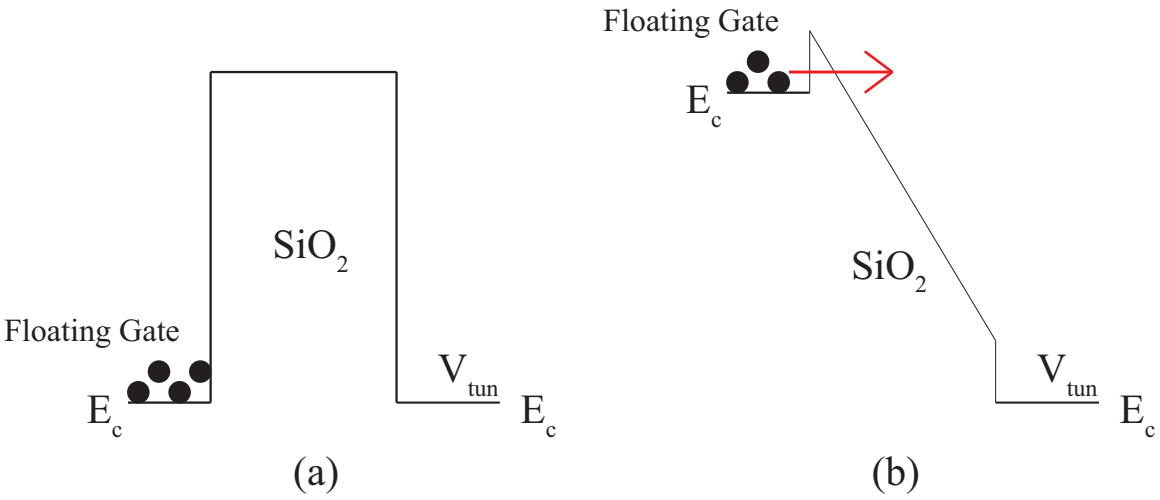


Figure 4. Band diagram of electron tunneling.

band. Fowler-Nordheim tunneling is the procedure commonly used to induce this phenomena. For a reasonably sized oxide insulator, illustrated in Figure 4a, the barrier is enough to prevent conduction under normal conditions. By applying a large field across the tunneling capacitor, the bands are bent so steeply that the electrons see a thin enough barrier that they can pass through it, as in Figure 4b. The ultimate result of this tunneling is a decrease in electrons on the floating node and thus a raising of V_{offset} in Equation 3, the effective gate voltage.

Hot-electron injection is the process by which electrons are added back to the floating gate, a diagram of which is in Figure 5. This injection process is performed by creating high drain-source and gate-source potentials. The gate potential creates a channel and the drain potential creates a high field in the device. Under these conditions, when a minority carrier enters the channel, it is accelerated with high energy toward the drain. When this carrier collides with the drain, it impact ionizes and creates an electron-hole pair. At this stage, with the high field from the gate, some of these “hot” electrons have enough energy to pass through the oxide to the gate region. The net effect of this is an addition of negative charge to the floating gate, lowering the effective gate voltage (V_{offset}).

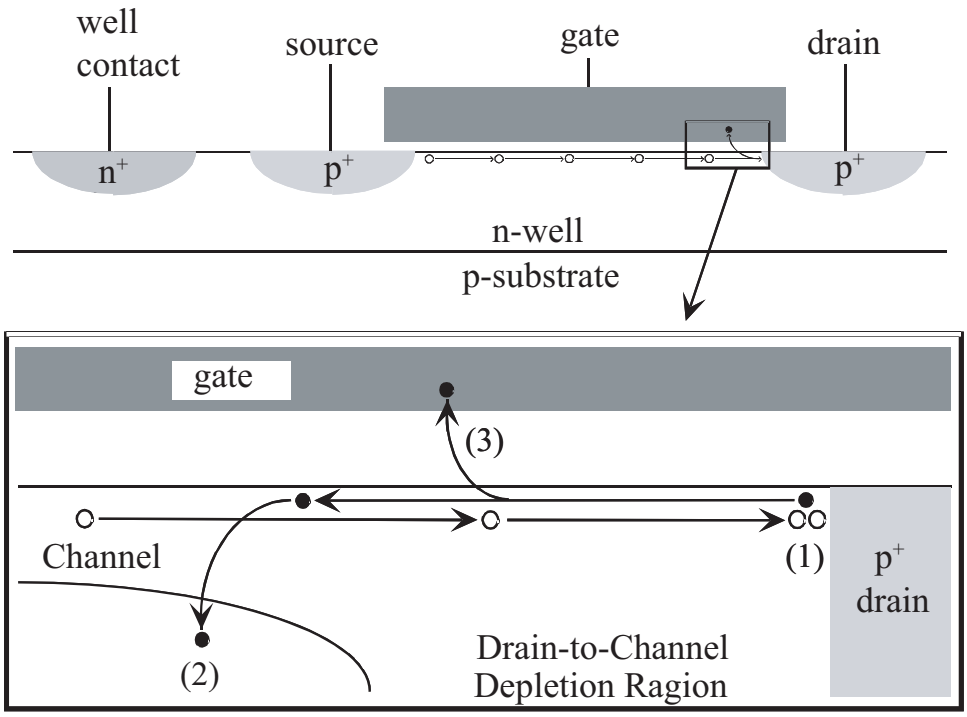


Figure 5. Diagram of hot-electron injection.

2.3 Array Programming

In order to rapidly program arrays of multiple devices, a selection procedure has been developed [3]. By arranging the transistors in a two-dimensional array, such as that in Figure 6, we can selectively target a particular device based on its row and column address. This two-element addressing lends itself nicely to the two-parameter injection. By tying all of the gates of a particular dimension together and the drains of the other dimension together, then applying an appropriate gate/drain voltage to the desired row/column, only one element will be under the right conditions for injection. Tunneling, however, only involves one parameter, namely the voltage coupled in across the MOS capacitor. In this capacity, tunneling is used as a global erase, whereas injection is used to program particular elements.

Figure 7 shows the floating-gate matrix element in more detail and the incorporation of indirect programming [4]. In this Figure, the transistor on the left is in the circuit path, and

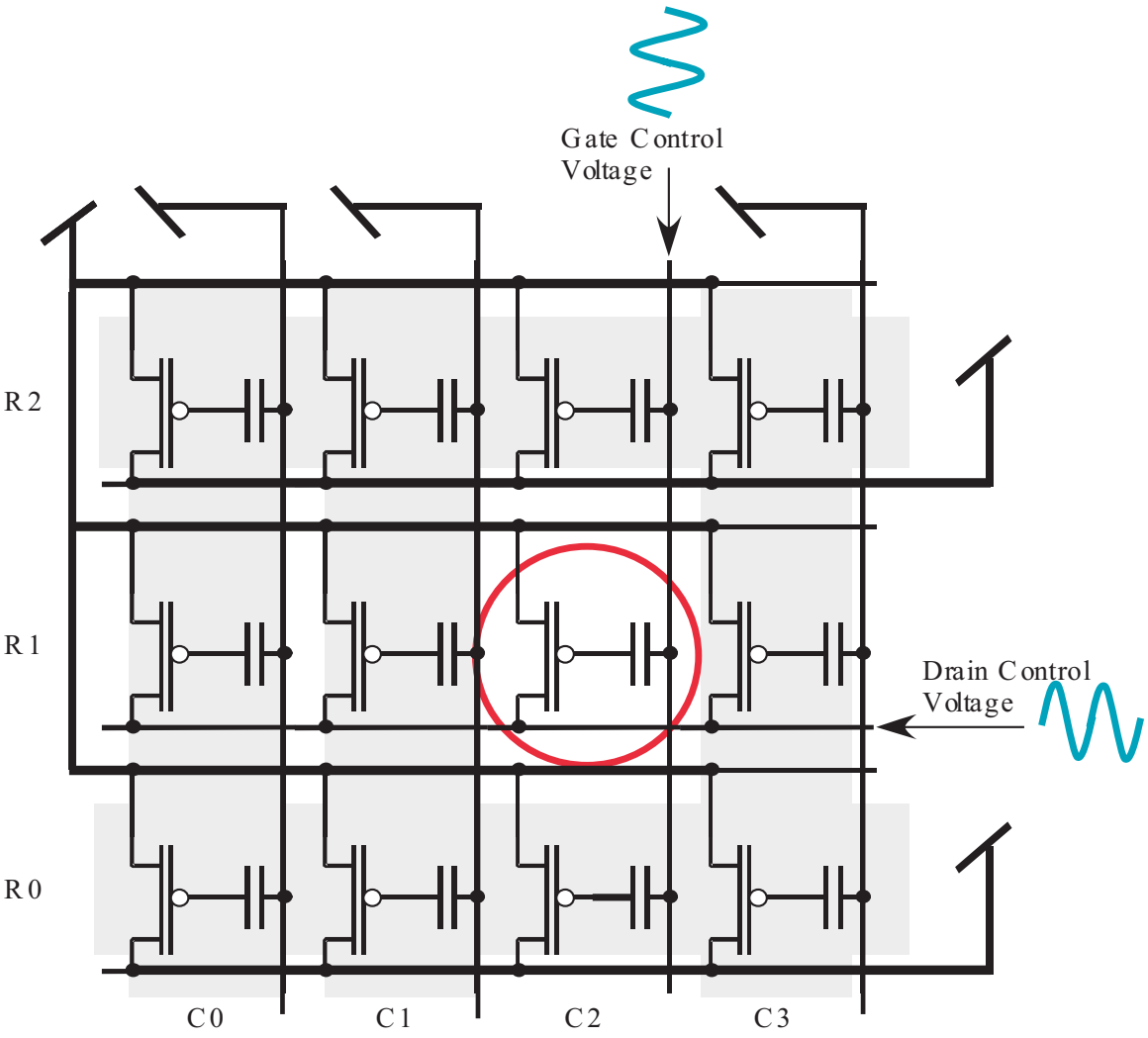


Figure 6. Array isolation.

sharing its floating gate, the one on the right is connected to the programming structure. If directly programmed, a floating-gate transistor needs to be disconnected from the circuit to have proper control over its source and drain voltages. To disconnect, a 2-to-1 mux is needed for each floating gate. The addition of this mux increases the overall switch count, and, thus, parasitics in the signal path.

By using indirect programming, the in-circuit transistor does not have to be disconnected, the other pFET is tied to the drain line and the select circuitry. Now, injection can be performed on the other pFET, with the resulting charge being deposited on the common

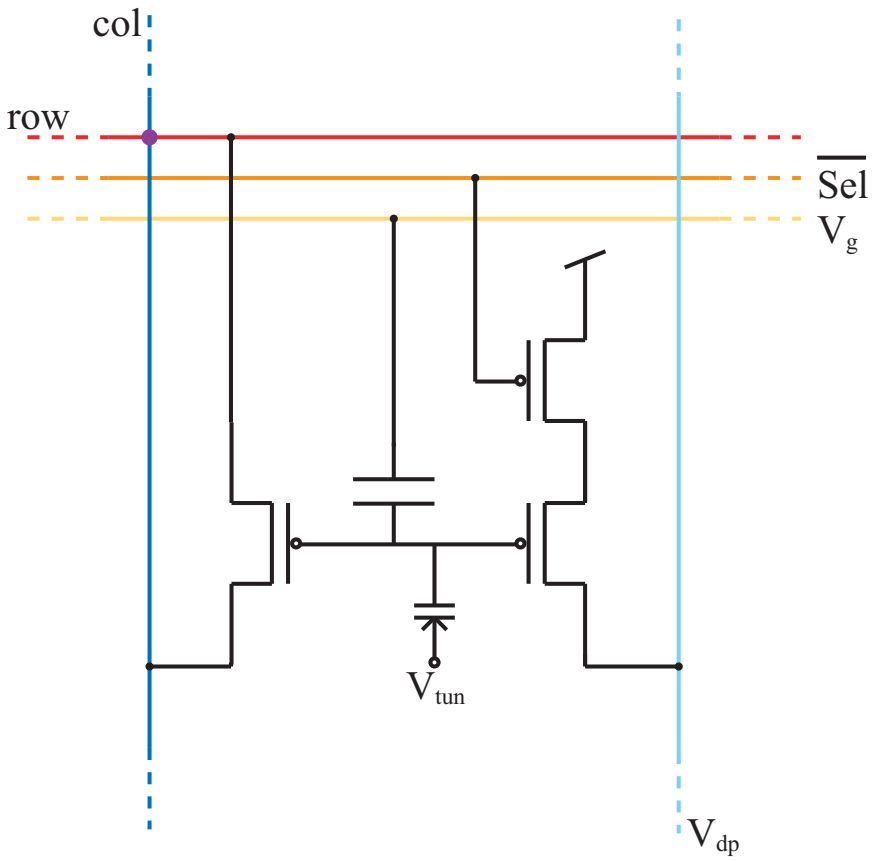


Figure 7. Floating-gate switch.

floating gate, without the need for any disconnection circuitry. In addition to decreasing the parasitics, this also decreases the area of the cell and increases the programming speed.

CHAPTER 3

FIELD-PROGRAMMABLE ANALOG ARRAY

3.1 The FPAA Advantage

The traditional design flow for a custom analog IC is shown in Figure 8a. In this typical process, a design is created and simulated as much as possible before it is sent off for fabrication. This fabrication step can take months and is very expensive. With the IC in hand, a further round of testing often reveals a level of performance not quite up to what was predicted by the simulator. This results in more iterations of the whole process, including the time consuming and costly fabrication step before an acceptable circuit is produced. Figure 8b shows the design flow for an analog system developed on an FPAA. After the initial simulation, we are able to synthesize and test the design on real hardware in a matter of minutes. By testing a physical chip, we can see all of the non-idealities that the simulation models often do not display. This allows us to iterate between simulation and hardware testing, leaving a single fabrication run as the final step. This FPAA design flow stands to drastically reduce a design team's fabrication cost and time to market. In addition to being a testing tool, FPAAs can also be used as the final product, much like an FPGA, completely eliminating the costly fabrication step.

3.2 FPAA Topologies

FPAAs are composed of two essential parts: computational elements and interconnects. In most topologies, the computational elements are arranged in Computational Analog Blocks (CABs) and the interconnects are an arrangement of switches that connect the wires running into and out of these CABs. As more researchers are discovering the importance of programmable analog, the questions of how to implement the switches and to what granularity to target are becoming more important.

An early FPAA work was Gulak's, in which he used "switch blocks" of cross-bar

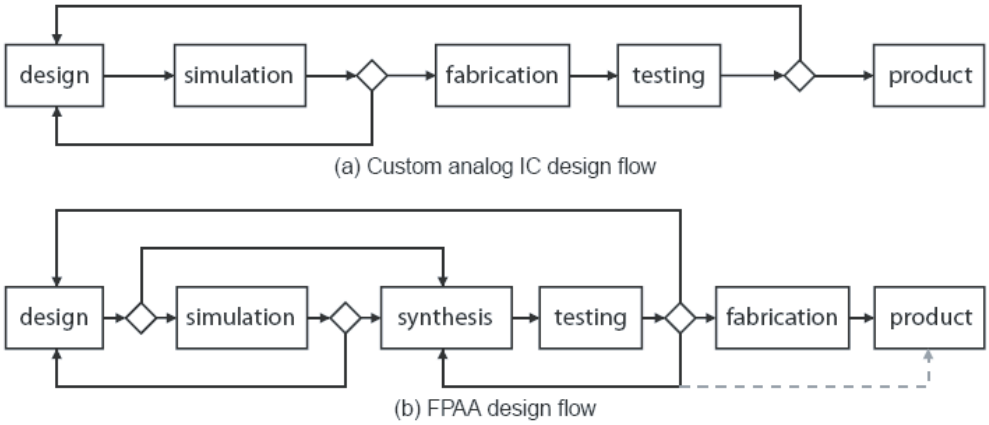


Figure 8. FPAA design flow. (a) Traditional analog design flow. (b) FPAA design flow.

switches and a shift register to control the connections between CABs [5]. The CABs in his design were arranged so that they could only implement one of seven limited functions (variations of comparing and multiplying), which was determined by a three-bit code applied to it. Since then there have been several topologies with CABs based on switched capacitors [6, 7, 8] and switched currents [9], but these FPAAs are mainly restricted to implementing discrete time filters. Other topologies that have been reported include CABs based solely on G_m [10] and op-amp [11] cells. These, however, can only synthesize a small subset of possible designs with their limited components. The architecture that shows the most potential for developing large scale analog systems is the RASP family of FPAAs [12]. The RASP FPAA provides the granularity for implementing the largest variety of circuits, with CABs containing basic analog elements such as FETs, OTAs, and capacitors. An additional superiority of the RASP is its use of floating-gate switches as opposed to the other topologies' use of T-gates controlled by a memory bank.

Figure 9a shows how a typical cross-bar switch matrix is arranged. There is a horizontal line and a vertical line, and where they cross is a switch, which, in this case, is a T-gate. This is a logical choice for implementing a FPAA switch matrix, because it is the structure commonly used by FPGAs. To create a switch this way involves a programmable memory

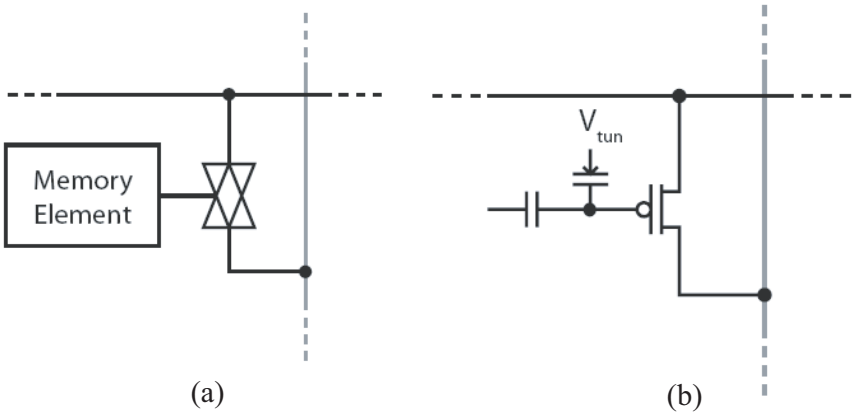


Figure 9. Switch implementations. (a) Transmission gate as switch element. (b) Floating gate as a switch.

bank that can select which switch is open and which is closed.

By using the floating gates discussed in Chapter 2, we can replace the T-gate and memory element by a single device, which is shown in Figure 9b. This drastically reduces the real estate required to create a switch. By using the tunneling and injection techniques of Chapter 2, we can program the switch to be open or closed. Referring back to Equation 3, we can see that the effective threshold can be pushed all the way high or all the way low depending on the amount of charge on the floating node. By moving the threshold, we create a switch with a more constant conductance range than a T-gate, as demonstrated in Figure 10. The T-gate is shown to pass a good low and a good high due to the combined efforts of the nFET and pFET (the pFET contribution is also shown in the Figure). However, its mid-rail resistance is very non-linear. The FGpFET, on the other hand, shows a much more steady switch resistance because we are effectively pushing its threshold very high or very low.

A fortunate bonus to using floating-gate elements in the switch matrix is that they can also be used for computation [13], which is detailed in Chapter 4. Since we can program the threshold to any value, we are not limited to simply on or off conductances. This is what gives floating gates a major edge in analog switch matrices, because since the switch fabric takes up a large portion of the system anyway, we are basically getting a free source

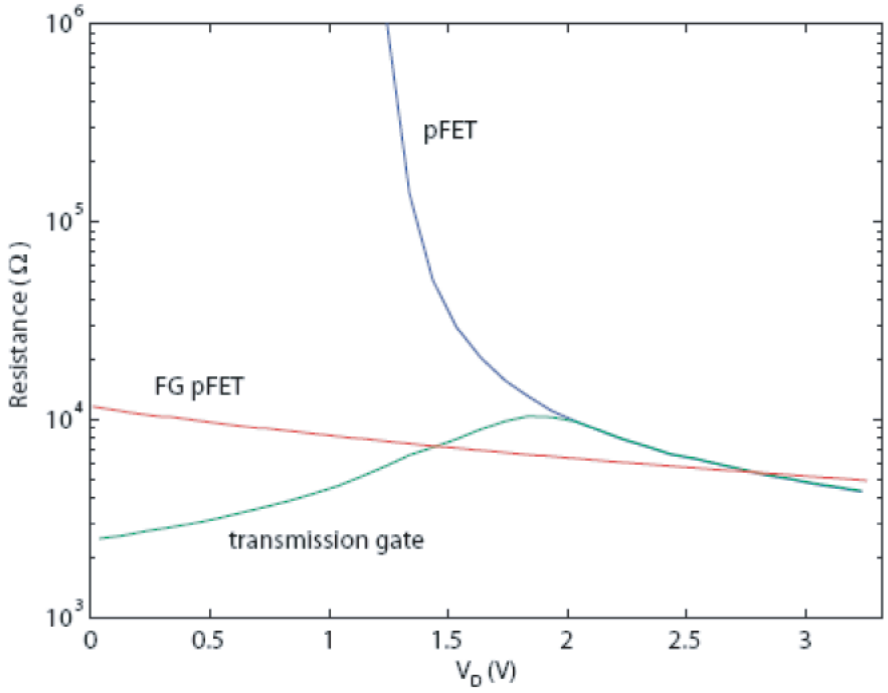


Figure 10. Switch resistance. Compares the conductance of three types of switches as functions of the voltage across it.

of analog memory, programmable biases, and multiplier weights.

3.3 Reconfigurable Analog Signal Processor

The Reconfigurable Analog Signal Processor (RASP) family of FPAAs has been developed by the CADSP research group over the last several years [14, 15]. The fundamental technology of the RASP line of FPAAs is the use of the floating-gate switch matrices outlined in the previous Section. With this floating-gate switch matrix as a framework, there are an unlimited variety of applications that a RASP FPAA can be geared to, defined by the composition of its CABs. For instance, there is a general-purpose FPAA (2.Xa) which has standard analog components in the CABs, a bio-FPAA with neuron channel models in the CABs and MITE-FPAAs [16] with translinear loops, with the potential for countless others. In this Section, I describe the 2.8a and 2.9a general-purpose FPAAs as well as the 2.9b high-speed FPAA.

3.3.1 RASP 2.8a

RASP 2.8a is the general-purpose FPAA of the 2.8 line [17]; the layout is shown in Figure 12 and some relevant parameters are given in Table 1. It was fabricated in a .35μm double-poly CMOS process through TSMC. With a size of $3mm \times 3mm$, it was able to accommodate 32 CABs. These CABs contain a variety of basic analog components such as: nFETs, pFETs, $500fF$ capacitors, 9-transistor OTAs, floating-gate input OTAs, MITEs, and T-gates. In addition to the CAB components, there are fifty thousand floating-gate switches that can be used for computation. One advancement in this line of FPAA is the use of nearest-neighbor routing, which as an alternative to the existing horizontal/vertical globals and locals, creates direct connections to the nearest CAB to the left, right, top and bottom. A system of bridge switches was also introduced. This allows for more lines to be drawn as locals and then bridged to the top and bottom local lines if needed. Table 2 shows that by using these shorter connections, the line capacitance is greatly reduced.

Another advantage presented by the RASP 2.8, and above, is the incorporation of on-chip programming [18], the structure for which is shown in Figure 11. Moving the programming on-chip has allowed for much higher speed operation. The major contributor to this increase in speed comes from the floating-point current ADC. This is a ramp ADC with an adaptive logarithmic I-V converter on the front end, which allows for conversions

Table 1. RASP 2.8a Parameters

Process	$0.35\mu m$
Die Size	$3mm \times 3mm$
Power Supply	$2.4V$
Injection Vdd	$5.6V$
Number of CABs	32
Switch programming time	$N_{rows} \times 100\mu s$
Bias programming time	5 ms/element
Programming accuracy and range	9 bits over 100fA to 10 μA

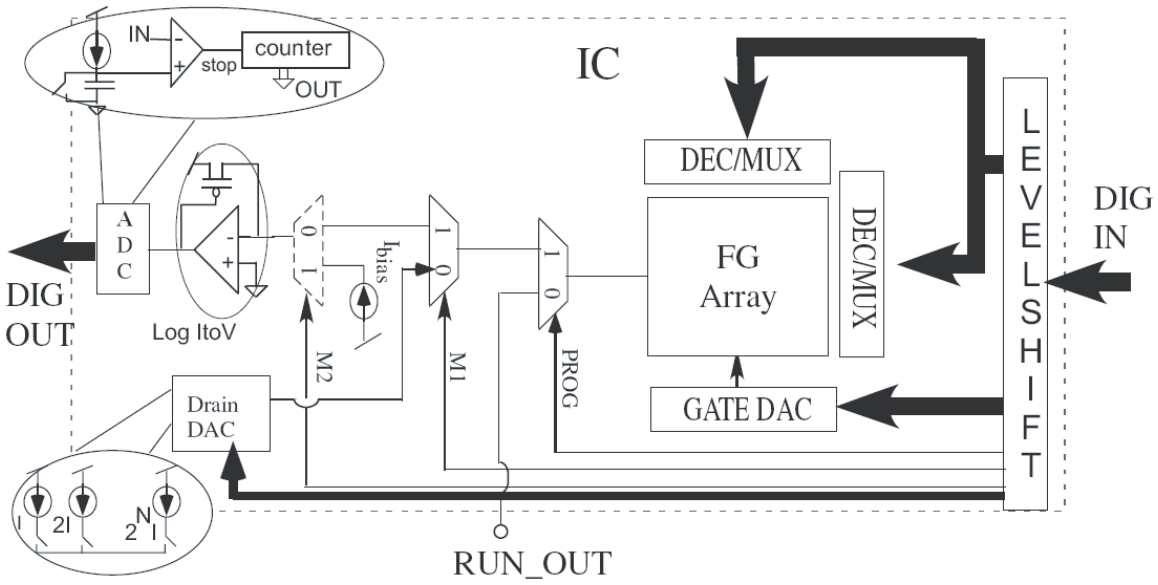


Figure 11. On-chip programming.

of seven decades of current in 200μs, which is much faster than off-chip measurements would take. Also on-chip are DACs for setting the gate and drain voltages during injection. When precisely programming (injecting) a device, the present current is measured and digitized with the I-V ADC. This voltage reading is then sent off-chip to a microcontroller and used to calculate how many and what size pulses are needed to hit the target current. These values are then passed back to the chip's shift register through an SPI interface. This register is used to control the selection lines as well as the gate/drain DACs. In order to control all of the off-chip aspects of the programming, as well as for testing, a custom Printed Circuit Board (PCB) was built, which is discussed in greater detail in Chapter 5.

Table 2. Line Capacitance

Nearest neighbor vertical	151fF
Nearest neighbor horizontal	228fF
Global	763fF

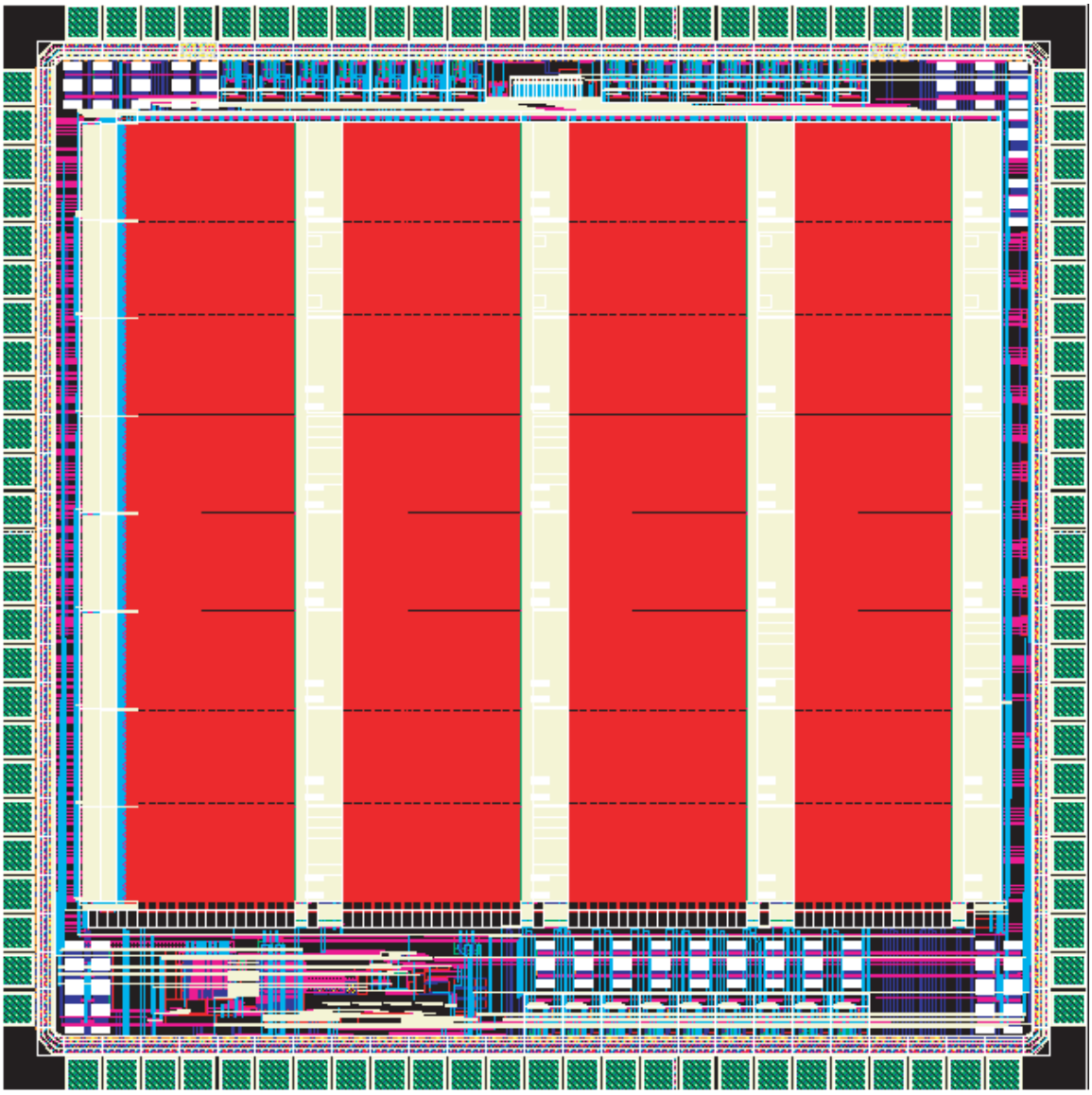


Figure 12. RASP 2.8a layout.

3.3.2 RASP 2.9a

The RASP 2.9a general-purpose FPAA, the layout of which is in Figure 13, is very similar to RASP 2.8a with the main difference being the size. The CABs and the routing are the same, but with a size of $5mm \times 5mm$, space for 84 CABs was available. The extra CABs allow for larger systems to be implemented. Due to this chip's larger size, it has a 200-pin package, so an adapter board was created to make it pin-compatible with the 2.8 line. This adapter board lets it fit easily into the ZIF socket of the existing programming board. The main drawback of this is that the 108 I/O lines of the 2.9 are reduced to the 52 I/O lines of the 2.8 (the amount that the current board can support). This has not been a problem because the 52 I/O has been more than enough to handle all of the systems attempted so far. Also, to help reduce I/O needed for a system, the blocks described in Chapter 4 can be used to set constant voltage or current values on-chip. The main foreseeable difficulty would be in trying to implement larger dimension VMM where all of the inputs and outputs are needed to get off-chip. To handle this situation, the on-chip T-gates would need to be employed to multiplex the I/O.

3.3.3 RASP 2.9b

The RASP 2.9b, shown in Figure 14, is the reduced routing version of RASP 2.9. The design choices for this chip were made to decrease the line capacitance and increase the signal bandwidth in an effort to target higher speed applications such as RF processing.

The 2.9b chip is identical to 2.9a in composition and number of CABs, but to reduce the line delay, the routing was restricted to mostly local with the exception of four global lines per CAB stack column. This provides wires with less unnecessary length and thus less unnecessary capacitance, as illustrated in Table 2. When it is required to connect neighboring CABs, the bridge switches are still available to connect the local lines of the two CABs.

In addition to the simple bridge switches, six in each switch matrix were implemented with voltage buffers. These buffers are inserted in between each CAB in order to drive the

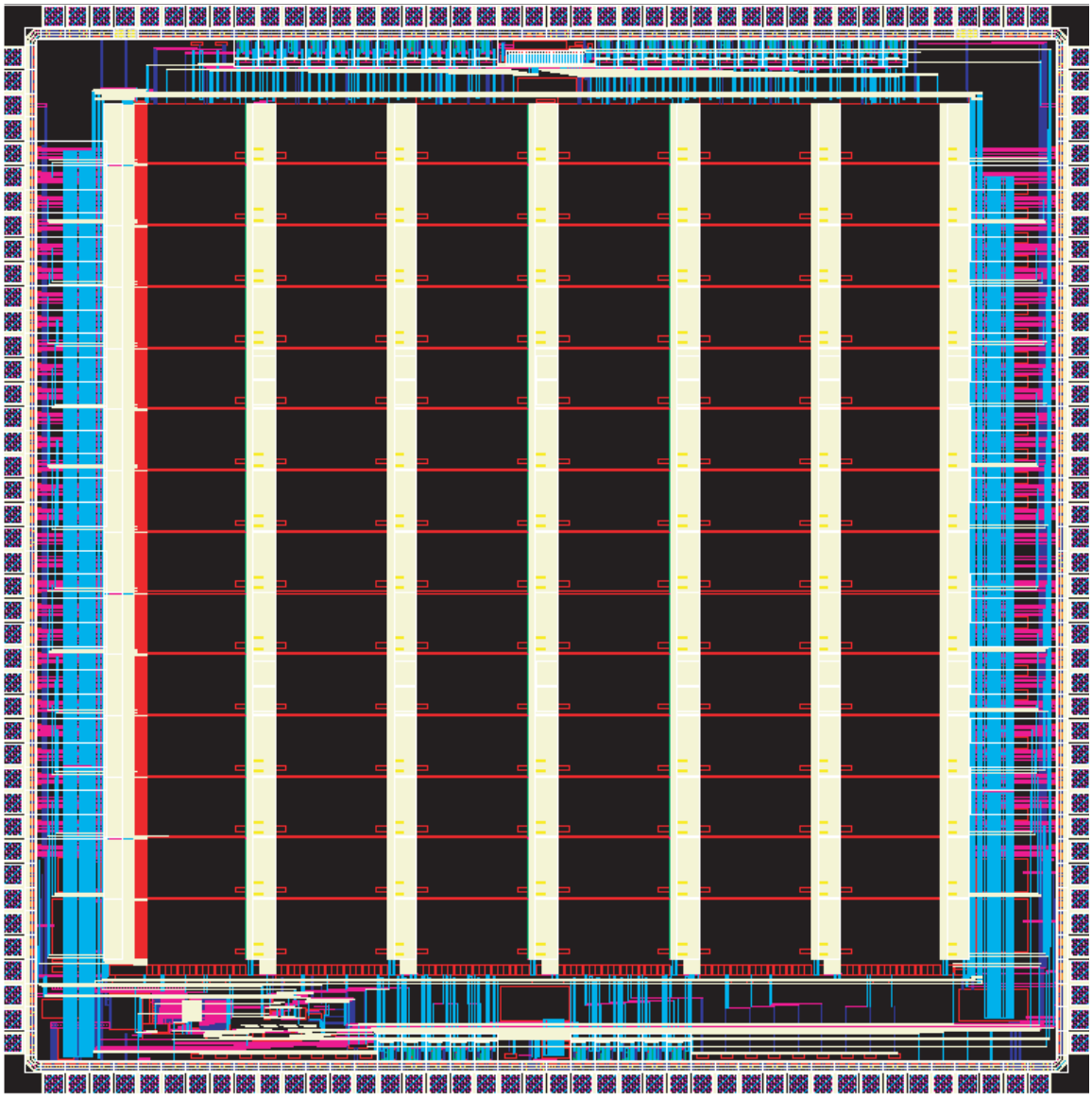


Figure 13. RASP 2.9a layout.

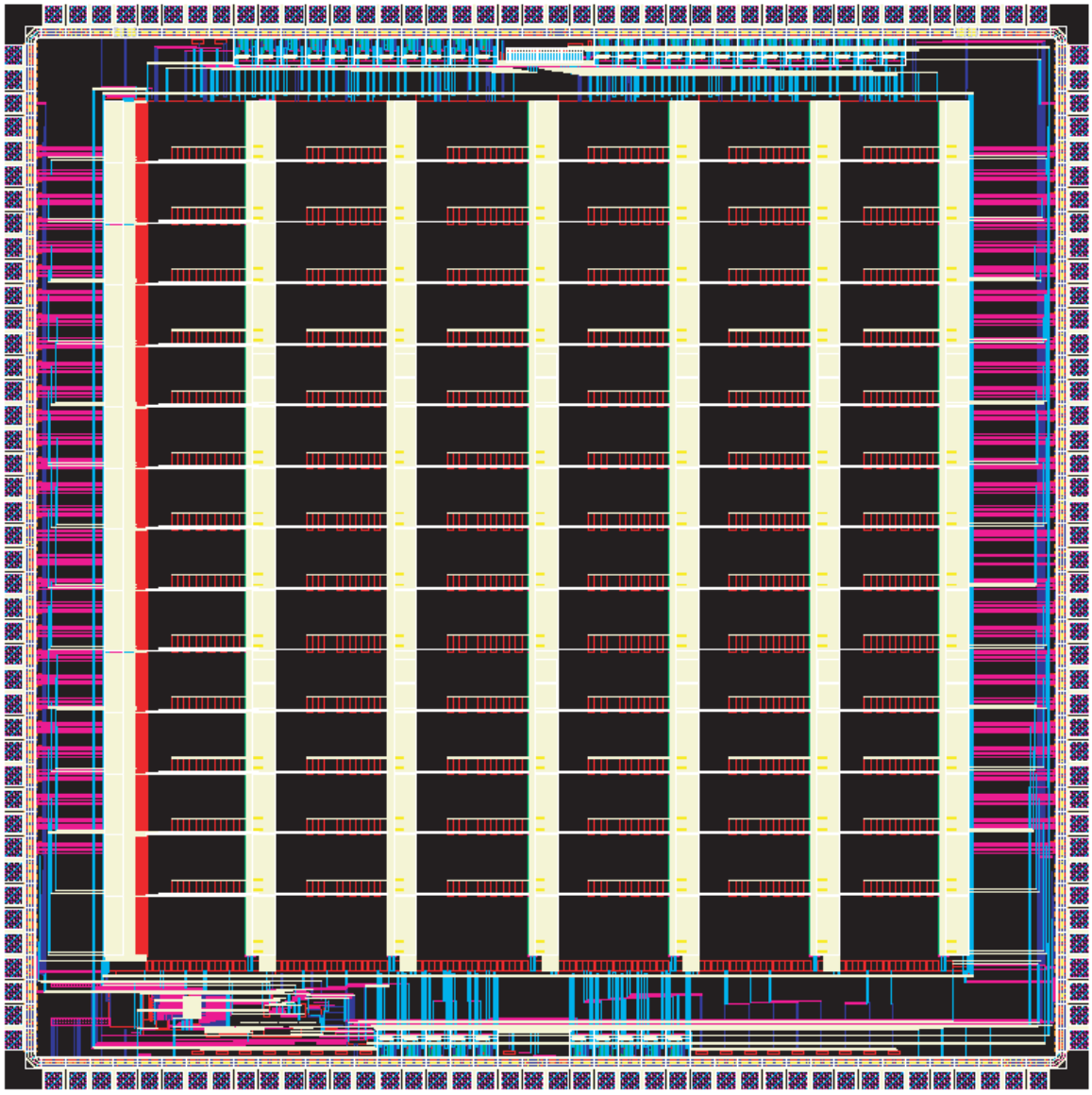


Figure 14. RASP 2.9b layout.

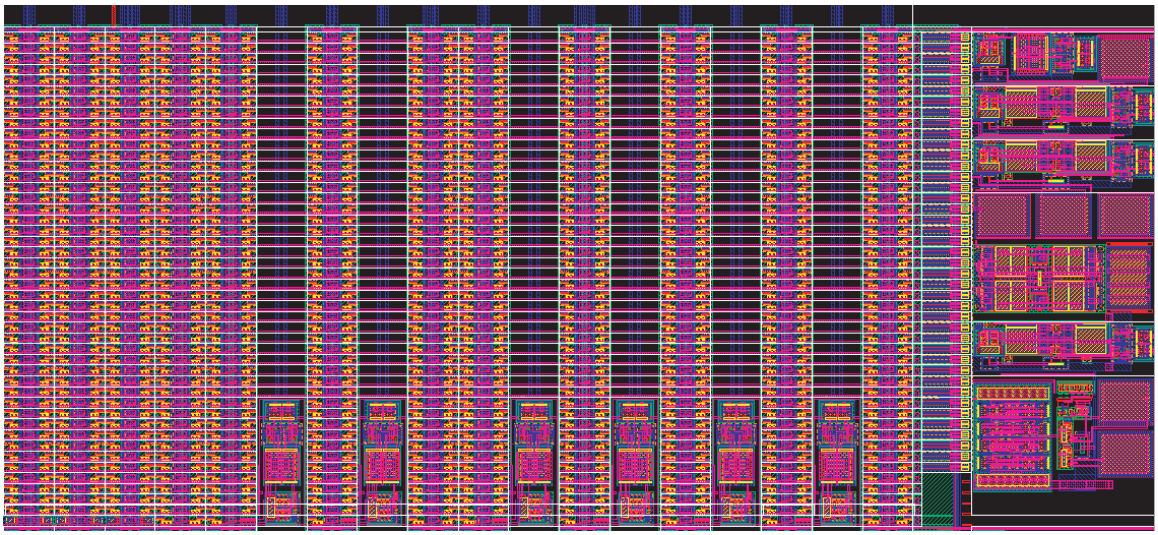


Figure 15. RASP 2.9b CAB layout.

capacitance on the next local line. Of the six, three are pointed up and three are pointed down to accommodate their unidirectional operation. These buffers can be observed in the layout of the switch matrix in Figure 15. Note that in order to make space for these buffers, six pairs of column switches were removed, resulting in a reduction of twelve vertical lines. This reduction is not a problem because routing density has not been an issue. Also, the 2.9a has as many lines as it did in the first place because it has a large variety of hard connected wires. The loss of this variety allows us to get by with less total lines.

In order to test this new FPAA, a special four-layer PCB is being built. This board is a stripped down version of the main board, with an emphasis on preserving I/O signal integrity. All of the nonessentials, such as programming infrastructure, will not be included on the board and wired over from the main board. To reduce noise and cancel electromagnetic interference, the signals will be jumpered with a twisted pair. The chip will be directly soldered down, without a ZIF socket. Off-chip drivers will be used to increase the signal strength to drive the pins. The pins will consist of twelve SMA connectors with the rest being standard headers. The I/O traces will be restricted to the top layer and made to be as short as possible and the thickness will be adjusted for proper impedance matching.

CHAPTER 4

ANALOG SIGNAL PROCESSING ON A FPAA

4.1 Analog Building Blocks

This Chapter introduces some basic analog building blocks and their FPAA implementation. The designs of these blocks are highly motivated by what is available in the RASP FPAA. For instance, since OTAs are readily available, they are used whenever possible; if a circuit calls for a programmed memory or constant value, a floating gate is used; and capacitors or low-conductance floating gates are used instead of resistors whenever possible.

When working in the analog space, it is very difficult to abstract a design. It is not as simple as digital design where one can merely describe a system as a whole in a hardware description language and a compiler can create the system. If this were so, many analog circuit designers would be out of a job. In an effort to make analog processing more accessible, we have compiled a library of commonly used analog building blocks. These blocks still need to be built by a knowledgeable circuit designer in the first place, but once created, they can easily be connected together and utilized by someone whose expertise lies elsewhere.

Obviously, with this approach, not all of the design decisions typically available to a circuit designer are presented to the end user. For instance, transistor sizing would not be an appropriate parameter because the transistors of the FPAA are fixed, but the bias current of an OTA that sets the corner of a filter would be parameterized for the user.

In this Chapter, I present a small sample of the countless possibilities for analog blocks. In an effort to demonstrate the variety of possibilities, three types of processing blocks are described: elements for computation (WTA, summer, VMM, MITE), sources of constant values (voltage source and current source), and stages for interfacing with the off-chip world (V-to-I, I-to-V). All of the blocks here have been built into the Simulink library discussed in Chapter 5, and all of the data is from the RASP 2.8a FPAA.

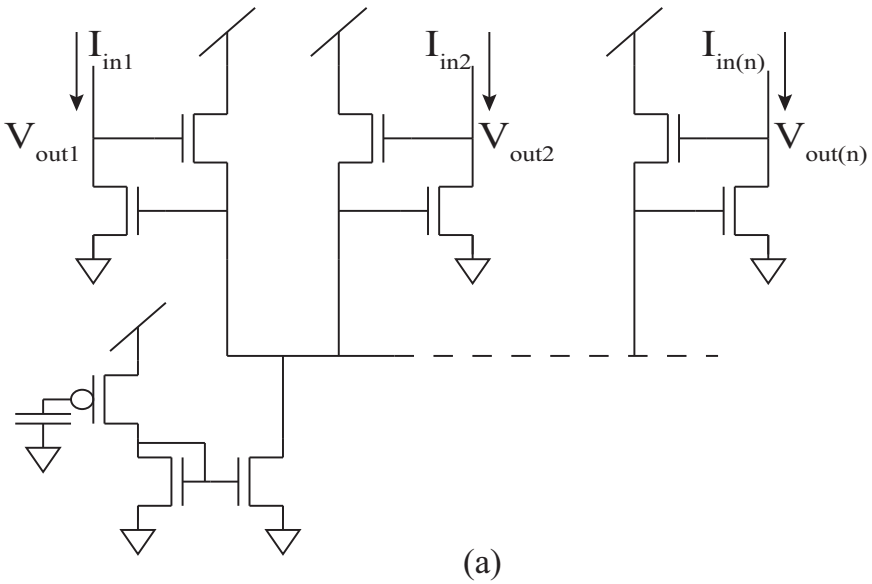
4.1.1 Winner-Take-All

A simple yet powerful signal processing block that finds many uses, particularly in neural networks, is the winner-take-all (WTA) [19]. The WTA is a current-in, voltage-out device, shown in Figure 16a. The currents come in on the drain of a nFET transistor and are logarithmically translated into a gate voltage; as the current increases, its gate voltage will also increase. With all of the gates of the input transistors tied together, when one raises, they will all be forced to raise with it. Now, the other transistors with lesser input current will have the gate voltage corresponding to a higher current. Since they all have a common source, the only way to compensate for this is for their drains to decrease. These drains are all connected to the inputs of diff-pair, so when one is higher than the rest, it will command all of the bias current. This creates a feedback loop forcing the voltage of the winning branch to a discrete high value and all the others to a discrete low value. This operation can be extended to order *n* by adding more branches on the common node. In terms of an FPAA implementation, it is important to note that the tail bias current can be programmed with a floating-gate device.

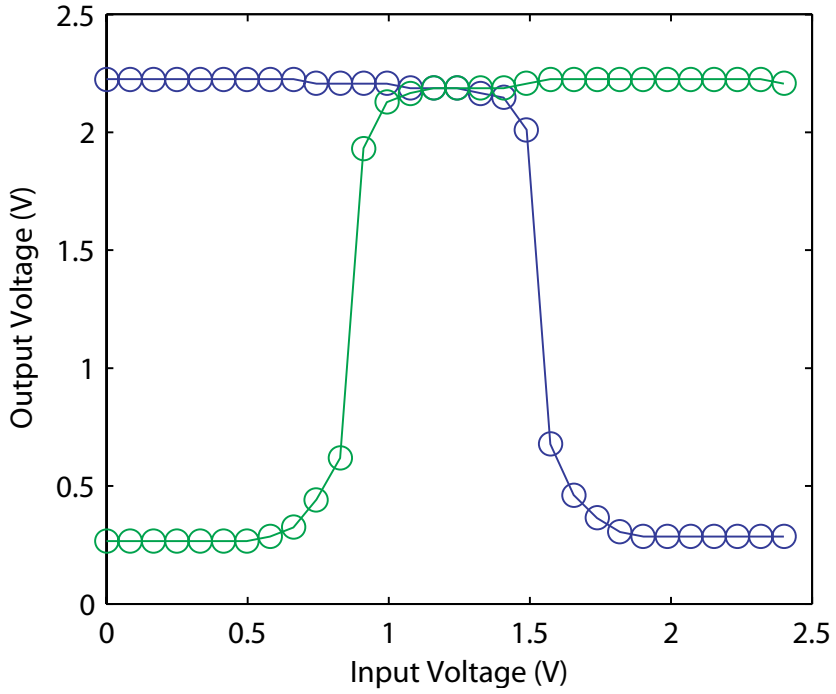
Figure 16b shows the output of a 2-dimensional WTA. In order to test the circuit with the DACs on our evaluation board, the input current is created by a control voltage on a pFET current source. The x-axis shows the voltage on input 1, the blue curve, with the voltage on input 2, the green curve, held at 1.2V. Input 1 is initially “winning” because with a lower voltage on the pFET gate, it has a higher current. When this input voltage crosses the 1.2V threshold, input 2 becomes the winner, as indicated by the transition of output 1 from low to high and output 2 from high to low.

4.1.2 Capacitive Summing Block

In signal processing, a very important function is the ability to add signals together. Current-mode signals are easily summed by KCL and voltage-mode signals would ideally be summed by an op-amp with resistive feedback [20]. However, due to their physical size and variability, resistors are used sparingly on most ICs and not included in the CABs of the FPAA.



(a)



(b)

Figure 16. Winner-take-all. (a) The WTA schematic of order *n*. (b) Output characteristic of a 2-dimension WTA.

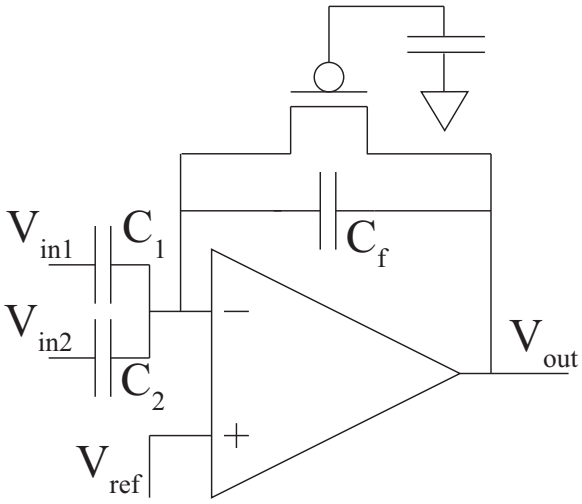


Figure 17. Capacitive summing block.

Fortunately, we are able to use the capacitors and OTAs in the CABs to achieve the same effect. The capacitive summer is shown in Figure 17. The inputs are coupled onto the inverting terminal of the amplifier through capacitor C_i, and the weighted summation is then given as

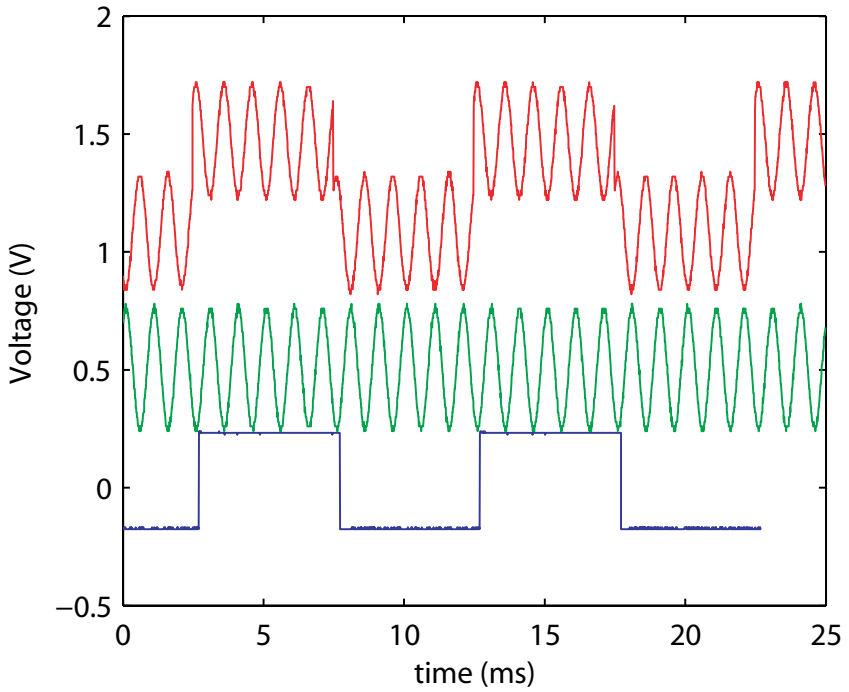
$$V_{out} = -\sum_{i=1}^{n}\left(\frac{C_i}{C_f}V_i\right) \tag{4}$$

where the weight on the i^{th} input is determined by the ratio of that input capacitor to the feedback capacitor. By allowing $C_i = C_j$, the sum will give equal weight across all inputs.

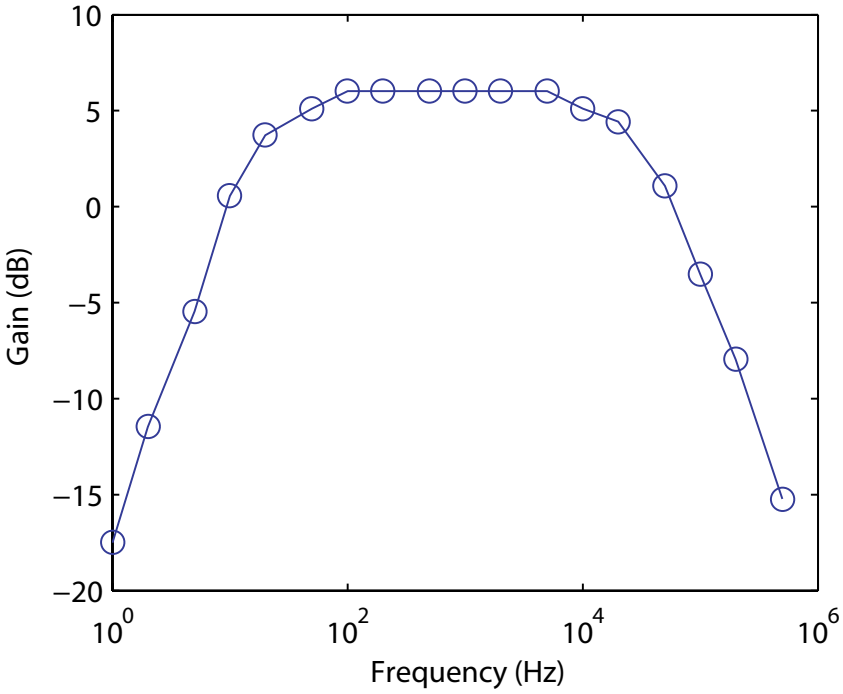
Figure 18a shows the transient response of the summation of two signals, a sine wave (middle) and a square wave (bottom), with $C_1 = C_2$. The resulting wave (top) is an equally weighted combination of the two. Figure 18b shows the frequency response of the same block, which reveals an operating range of 10–100KHz. For this plot, the same input signal was applied to input terminals 1 and 2, hence the gain is 6dB.

4.1.3 Input/Output Stages

In analog processing, signals are often computed in current-mode because they can be mirrored, scaled, and summed easily. However, because it is much more convenient to produce and read voltage signals with DACs and ADCs, it is important to have the ability



(a)



(b)

Figure 18. Capacitive summer output characteristic. (a) Shows the combination of two signals, a sine wave and a square wave. (b) The frequency response shows an operating range of 10–100KHz.

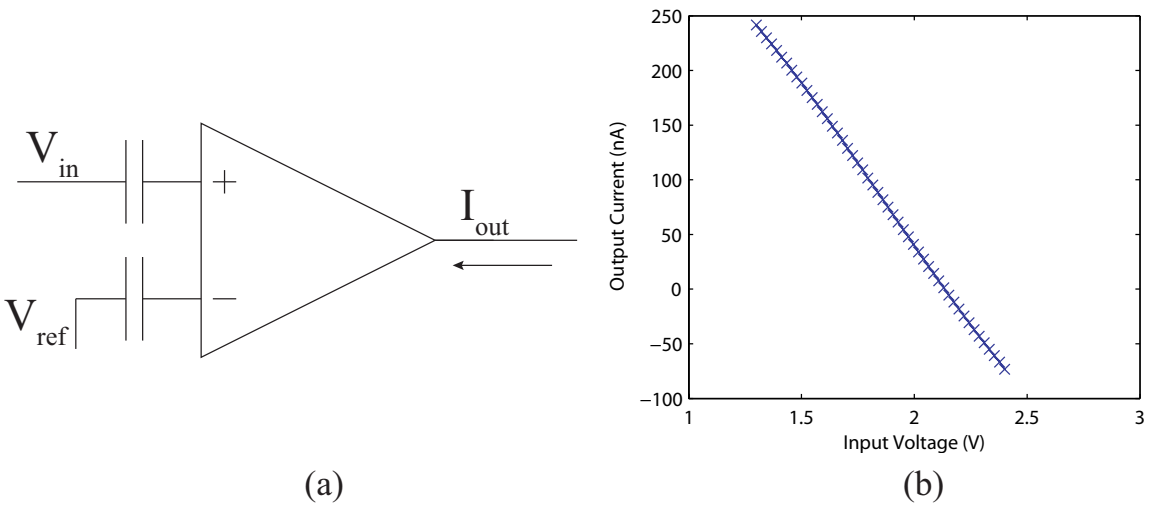


Figure 19. V-to-I input stage. (a) An ota configured as a V-to-I. (b) The output current as a function of input voltage.

to convert signals between the two modes.

Given the elements available on the RASP FPAA, the best choice for V-to-I conversion was a floating-gate input OTA in the simple configuration of Figure 19a. The floating-gate input OTA was chosen over a regular OTA because the input capacitors attenuate the input and give the amplifier a larger linear range. Figure 19b shows the output current as a function of input voltage. The gain on the OTA can be programmed to provide the desired amount of current for a given input signal.

One design for an I-to-V interface stage is the sense amp shown in Figure 20a. This output stage mirrors the incoming current across a resistor, which then allows the current to be measured as a voltage drop, providing a linear I-V relationship. In order to measure the often small currents of interest (on the order of 100's of nanoamps), a diode-connected nFET is connected at the source of the input stage of the current mirror, providing sufficient gain. The resistor here was chosen to be $5M\Omega$ to provide a reasonable range of output swing.

A more elegant and compact alternative to the sense amplifier is the transimpedance amplifier (TIA), shown in Figure 21a. In fact, this output stage is similar in operation to

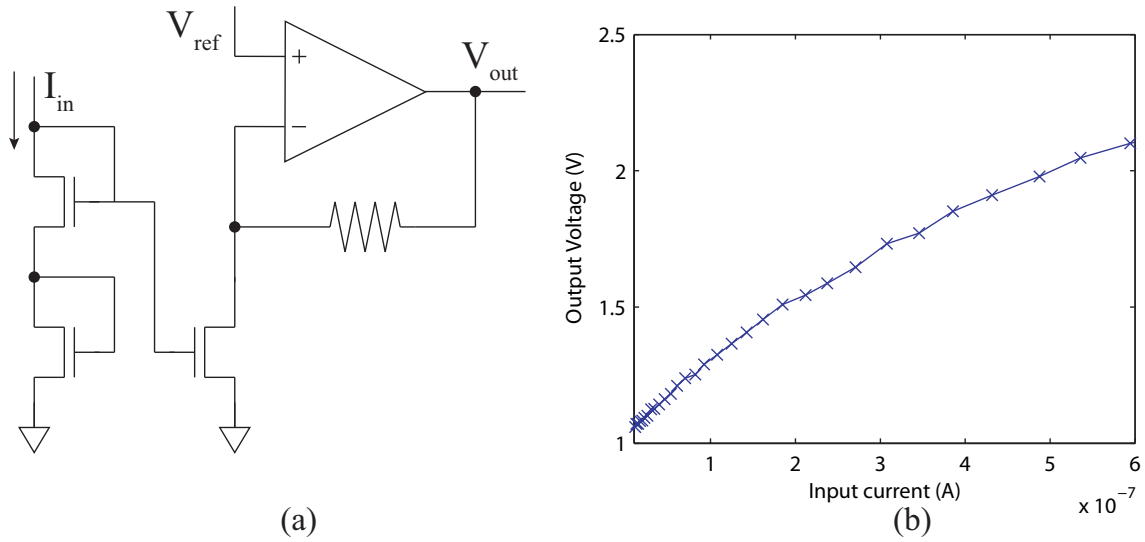


Figure 20. I-to-V output stage. (a) The sense amplifier creates a voltage representation of a current signal, which is desirable so the on-board ADC's can be used to read a response. (b) The output is a linear, wide-range voltage representation of the input current.

the sense amp, but since resistors are not included on-chip and thus need to be pinned out and used off-chip, a G_m stage is placed in the feedback path instead. This G_m block acts as a programmable resistance, making this circuit ideal in that it can be compiled all on-chip and is a very simple with few components. The output voltage is linear with input current, shown in Figure 21b, making it an ideal output stage. The voltage swing is large, 1.5V, allowing for as many as 150 steps with our on-board 8-bit ADC. The input current range can be easily tuned by the floating gate tail current of the G_m stage.

4.1.4 Programmable Voltage & Current Sources

In the analog systems with which we deal, constant voltages or currents are often needed. Constant voltages are used in such places as reference to an amplifier, and current sources are often used as biases, such as for the WTA. These could be generated off-chip, but we prefer to move as many parts of the system on-chip as possible, making the system more self contained and freeing up I/O lines.

A programmable voltage source is shown in Figure 22a. The design choices for this voltage source were highly motivated by the elements available in the FPAA's CABs, in this

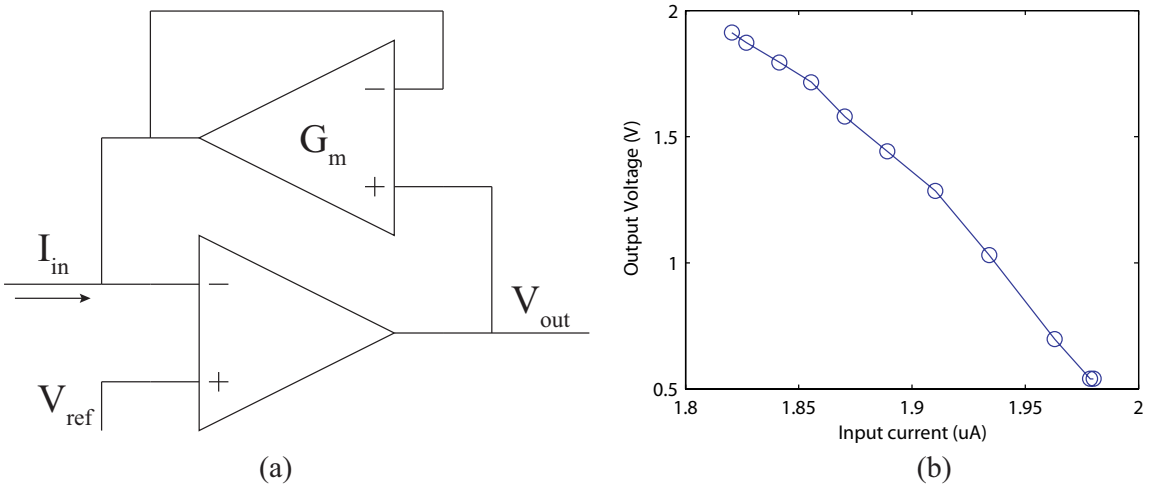


Figure 21. Transimpedance amplifier output stage. (a) The schematic for the TIA stage to convert the current to voltage. (b) The linear, wide range output characteristic of the TIA.

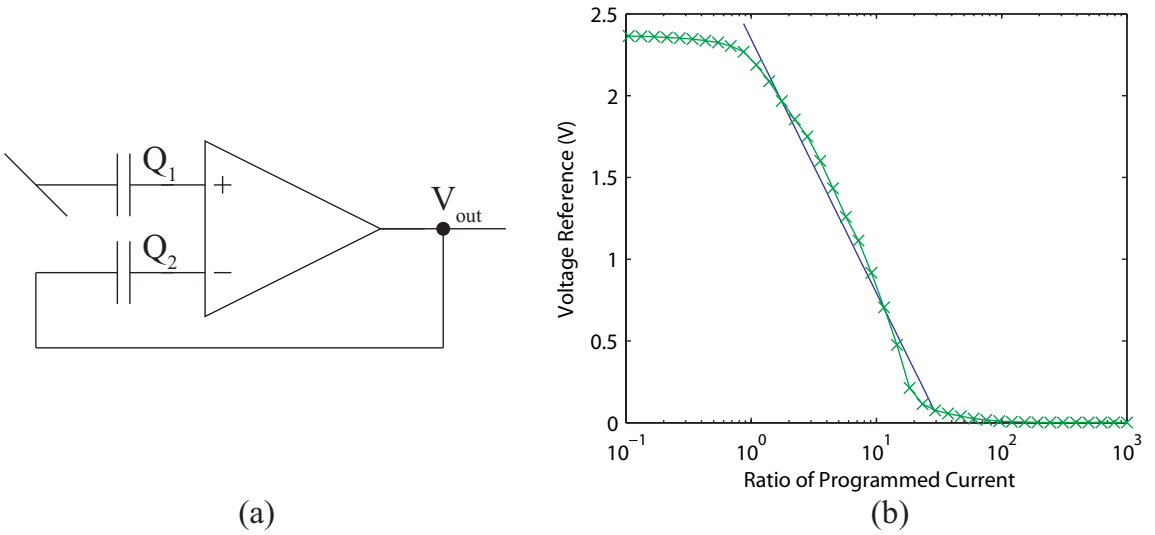


Figure 22. Programmable voltage source. (a) One of the programmable input OTA CAB elements, in a unity gain configuration, with the output voltage determined by the ratio of charge on the input capacitors. (b) The output voltage as a function of the ratio of programmed currents on the input floating-gate transistors.

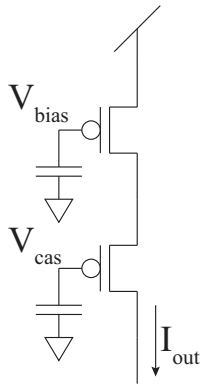
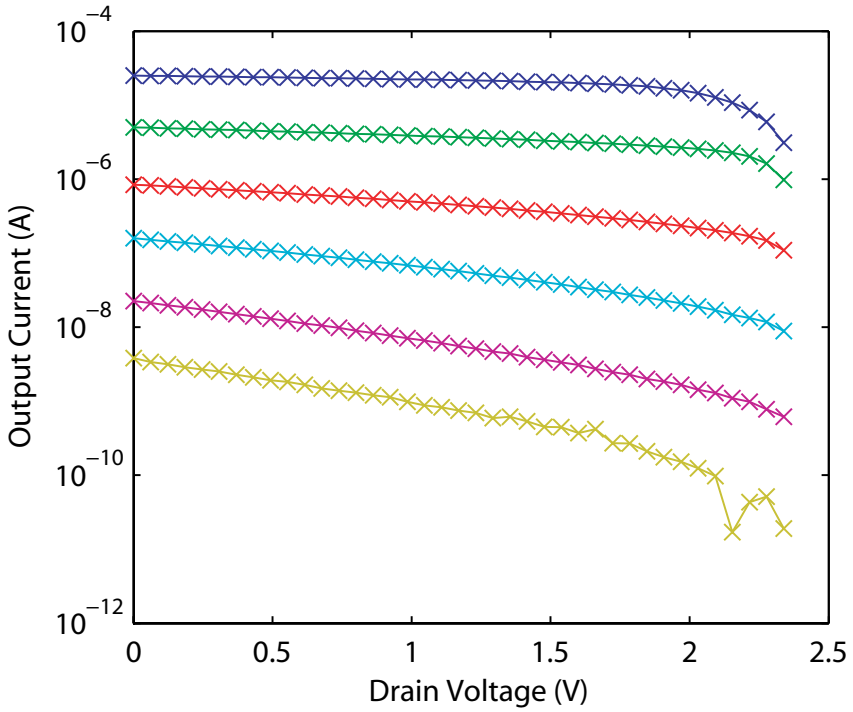


Figure 23. Programmable current source. Drawn with the output cascoded to increase the output resistance and reduce the current's dependence on the output voltage.

case a floating-gate input OTA. The OTA can be made to produce different constant output voltages by programming the inputs to varying ratios of currents, as shown in Figure 22b. Thus the voltage source allows us to eliminate off-chip resources while adding minimal complexity to the circuit.

In addition to voltage sources, the need for current sources is also encountered. For our purposes, it is convenient to implement the current source as a precisely programmed floating gate since they are of large supply in the FPAA switch fabric. Figure 23 shows the schematic of the programmable current source. It is extremely simple in its design—just one floating-gate element as the source and one on the drain as a cascode. The cascode is there to increase the output resistant and act as a current buffer, making the source more ideal by reducing its dependence on the output voltage. Figure 24a shows the output of the current source without the cascode, programmed to several values of current, as a function of the drain voltage. The increase in current with a decrease in drain voltage is noticeable and makes this source unideal. Figure 24b shows the output with the cascode, and it is much less dependent on the output voltage, making it more ideal.



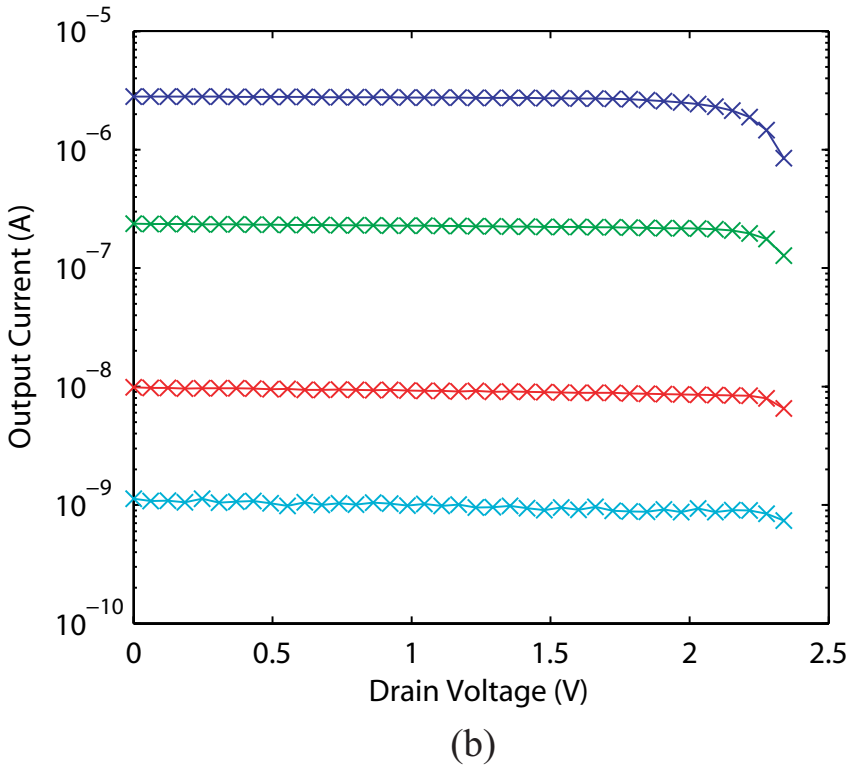


Figure 24. Programmable current source output characteristic. (a) The output current without cascode as a function of the drain voltage. (b) The output current with cascode as a function of the drain voltage.

4.2 Vector-Matrix Multiplier

The computation of vector-matrix multiplications (VMM) is an area in which an analog solution has excelled. A VMM can be used for countless signal processing algorithms, such as convolutions, 2-D image transformations, and FIR filtering. By performing this function in analog hardware, we can expect a large increase in speed and reduction in power, due to the highly parallel computation and subthreshold FET operation [21]. This makes an analog VMM ideal for real-time or mobile applications.

Although analog VMMs have been built in the past [22, 23], these have mainly been designed with a specific system in mind such as a CMOS imager. I present a VMM implementation on a FPAA, which will allow other engineers to prototype and incorporate a VMM solution into their own designs.

4.2.1 Vector-Matrix Multiplication

When performing a vector-matrix multiplication, we are solving the system $Ax = b$ where $A \in \Re^{m \times n}$, $x \in \Re^{n}$, and $b \in \Re^{m}$. For our purposes, it is enlightening to represent this relationship as the following.

$$b_i = \sum_{j=1}^{n} a_{ij} x_j, \ i = 1, ..., m \tag{5}$$

This can can be taken to mean that the j^{th} element of x must be multiplied the j^{th} element of the i^{th} row of A, then summed along that row to create the i^{th} element of b, making b a linear combination of the columns of A. So the vector-matrix multiplication can essentially be thought of as scaler multiplication and additions, which are ideal for implementing on our reconfigurable analog platform.

4.2.2 Structure of the VMM

In order to compute a function in the form of Equation 5, a multiplication and a summation are required. These operations can be carried out in a fairly straightforward manner on a RASP FPAA, as shown in Figure 25.

The VMM is fundamentally a current-input, current-output device, bearing a strong resemblance to a weighted current mirror. The input stage involves a log-compression conversion of the input current to a voltage by way of a floating-gate transistor and an OTA. This source voltage then sets the input voltage for a row of floating-gate switches, representing a column of the mathematical matrix, with each switch representing one single-quadrant multiplier. The multiplication factor is the ratio of current values that the multiplier floating gates are programmed to. The output currents of these floating-gate switches then depend exponentially on voltage and are summed for each input row to produce one output. This output is one element of the output vector b. The input-output relation is linear given that the multiplier acts as an exponential on the log-compressed input. Two- and four-quadrant multipliers may be constructed by using a differential signaling scheme for inputs and representing one multiplier with two or four floating-gate switches, respectively.

4.2.3 Output Characteristics

Since the inputs and outputs of the VMM structure are strictly positive currents, a differential scheme must be used for two- or four-quadrant multiplication. We define our differential currents as

$$I_{out} = I_{in+} - I_{in-} \tag{6}$$

where both I_{in+} and I_{in-} are positive values. So, for one four-quadrant differential multiplier element, we get

$$\begin{bmatrix} w_+ & w_- \\ w_- & w_+ \end{bmatrix} \begin{bmatrix} I_{in+} \\ I_{in-} \end{bmatrix} = \begin{bmatrix} I_{out+} \\ I_{out-} \end{bmatrix} \tag{7}$$

which, after multiplying the left hand side becomes the following.

$$\begin{bmatrix} I_{in+}w_+ + I_{in-}w_- \\ I_{in+}w_- + I_{in-}w_+ \end{bmatrix} = \begin{bmatrix} I_{out+} \\ I_{out-} \end{bmatrix} \tag{8}$$

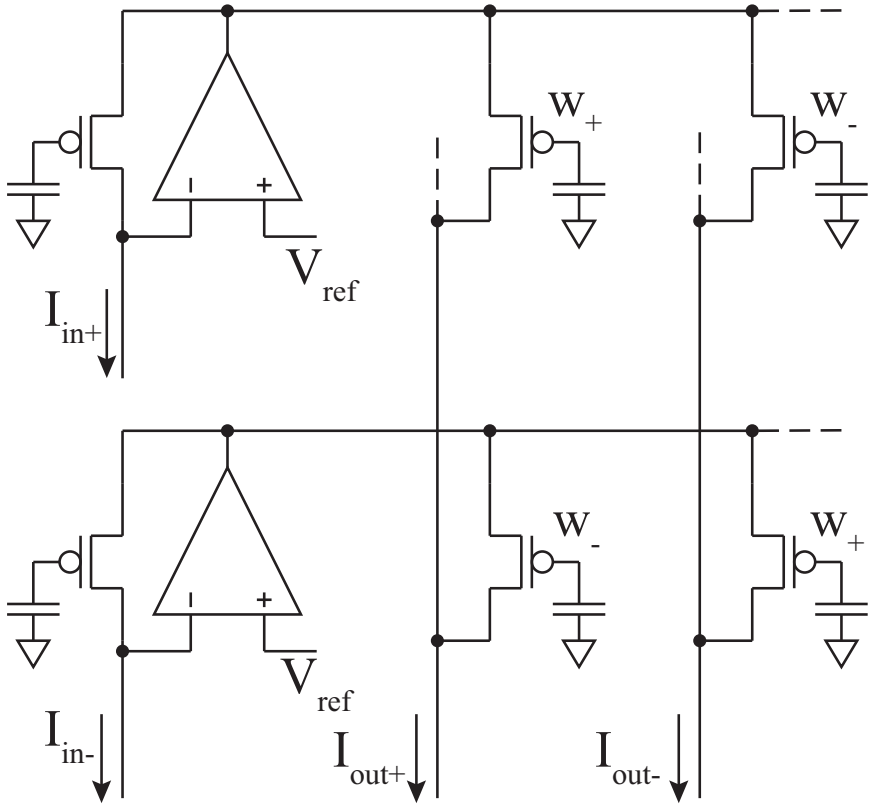


Figure 25. Vector-matrix multiplier. The input floating-gate stage produces a log compressed voltage representation of the input current. This is then broadcast to the floating-gate elements in each column of that row. The output floating gate produces a current which is a scalar multiple of the input. The currents are then summed along a column by KCL.

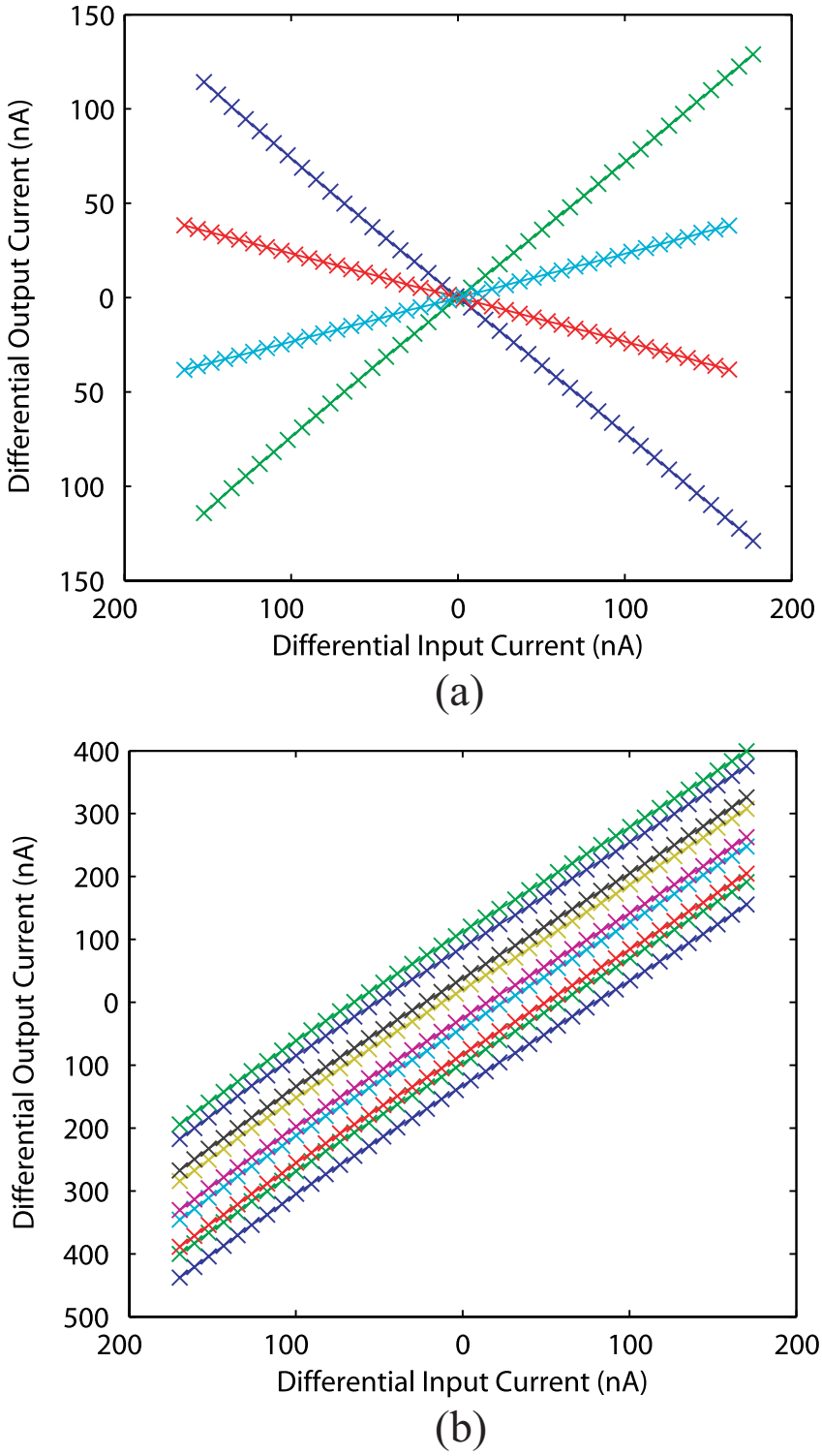


Figure 26. VMM output characteristics. (a) A 1x1 VMM. The inputs were swept differentially, to create differential outputs. The gains for this sweep were $\approx \pm 1, \pm 0.5$. (b) A 2×1 VMM. This sweep demonstrates the summation of the two inputs. One input is swept differentially for several constant values of the second input, which produces the vertical offsets.

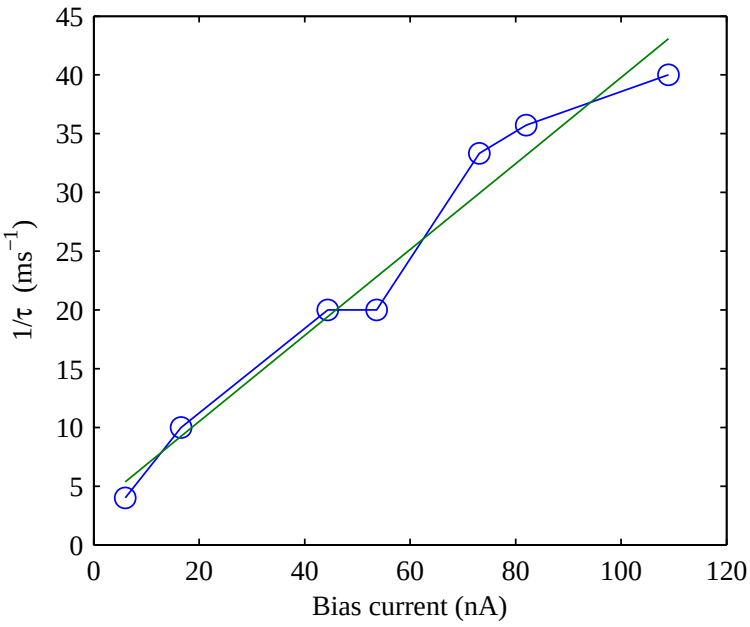


Figure 27. VMM time constant. The inverse of the time constant increases linearly with an increase in bias current. The slope of the rise is $(2.73pF)^{-1}$**, which corresponds to an overall capacitance of** $2.73pF$ **for the VMM.**

In order for the multiplication to remain symmetric, without deviating from the mean value of the differential currents, we define the weights as the following.

$$w_+ = |w| + \frac{w}{2},\ w_- = |w| - \frac{w}{2} \tag{9}$$

By tiling blocks of Equation 7, we can obtain a fully differential system, which is capable of four-quadrant multiplication.

The plot of a four-quadrant 1×1 multiplier is shown in Figure 26a, where the input in the form of Equation 6 is swept. Several positive and negative weights were programmed and the response is shown to be linear over the entire input range. A 2×1 multiplier is shown in Figure 26b to demonstrate the summation of the different rows of the input vector. For this demonstration, the first input is swept for several values of the second input, resulting in the vertical offsets.

4.2.4 VMM Time Constant

The frequency limitation of the VMM is primarily due to the time constants of the floating-gate switches. The time constant is exponentially dependent on bias current through the floating-gate multipliers, with lower time constants for higher bias currents. The time constants vary from $25\mu s$ for $109nA$ of bias current to $250\mu s$ for $6nA$ of bias current. Figure 27 shows a graph of τ^{-1} vs. I_{bias}. From this data, we found that the total capacitance of the VMM circuit used in this example was $2.73pF$. This is a reasonable value for total capacitance, taking into account the capacitances of two OTAs, two floating-gate switches, and line capacitances of routing within the FPAA.

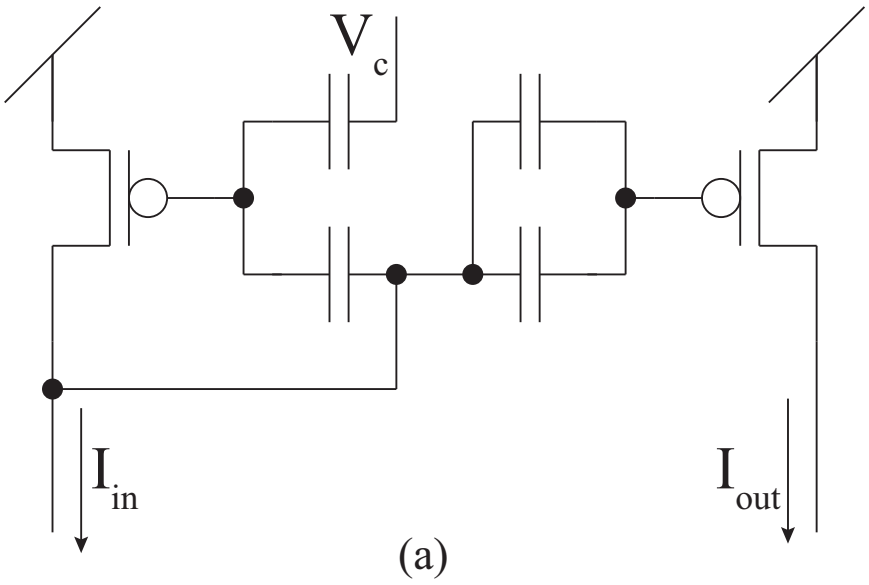
4.3 Multiple Input Translinear Element

Multiple-input translinear elements (MITEs), developed by Brad Minch, are circuit primitives that have more than one input and exhibit exponential current-voltage relationship [24]. For our purposes, it is easiest to implement this translinear relationship with a subthreshold MOS transistor and couple the multiple inputs on with a floating gate. This subthreshold MOS will now produce a current that is proportional to the exponential of the weighted sum of the coupled inputs, as shown in

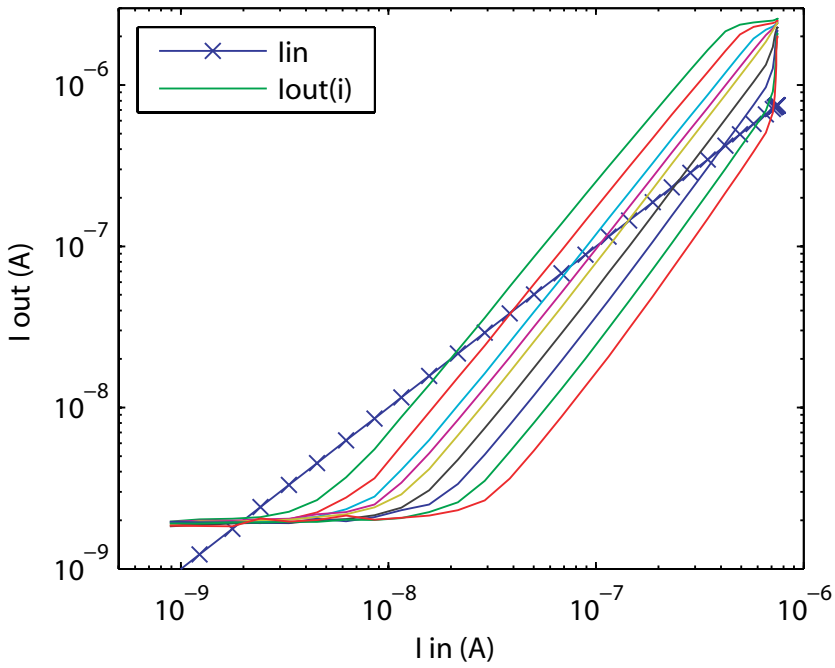
$$I = I_s e^{\frac{V_S - \kappa \sum w_i V_i}{U_T}} \tag{10}$$

where I_s is a pre-exponential scaling factor, V_s is the source voltage, κ is the capacitive division between the oxide capacitance and the depletion capacitance ($\frac{C_{ox}}{C_{ox}+C_{dep}}$), w_i is the input weight given by C_i/C_T , V_i is the input voltage, and $U_T = kT/q$ is the thermal voltage.

MITEs can be used for a variety of computation such as products, quotients, fixed power-law relationships, and ODEs. There are well-established synthesis techniques for MITE systems [25] and an entire FPAA has been built around them, making them an enticing tool for our analog signal processing library. In this Section, I briefly discuss an example MITE system, a squaring circuit implemented on both the RASP 2.8a FPAA and the MITE FPAA.



(a)



(b)

Figure 28. MITE CAB element. (a) Schematic of squaring function implemented with CAB-element MITES (b) Output of squaring circuit.

4.3.1 MITE CAB Elements

The RASP 2.8a FPAA includes four two-input pFETs per CAB, which if operated in sub-threshold, can be utilized as MITEs. For the squaring circuit, we need the weight between the input current and the output to be double. Since all of the coupling capacitors are the same size, this can be achieved by connecting one gate terminal of the input to two of the output, as shown in Figure 28a. V_c provides the reference to balance the units. Figure 28b shows the output of the CAB element squaring circuit. In the log-log plot, the output displays a slope twice that of the input, indicating a factor of 2 in the exponent. Several values of V_c were set to show the shift (but same slope) that the reference can produce.

4.3.2 MITE FPAA Implementation

The MITE FPAA was created by Dave Abramson [16]. This FPAA has a floating-gate switch matrix exactly like the other RASP FPAAs, but with MITEs in the CABs arranged in the translinear loops of Figure 29. In this configuration, each MITE has two gate terminals, which are connected to its neighbors in a loop. The gates are also wired out to the switch matrix; so depending on where the incoming signal enters then leaves the loop, a different computation will be performed. Figure 30a shows the squaring circuit implemented in these translinear loops. The output curve is shown in Figure 30b, where the bubbles are the experimental data and the red line is an ideal square. The data follows the ideal curve for a good part of the range, but diverges for higher currents. In Dave's Ph.D. dissertation, he attributes this divergence to kappa variation.

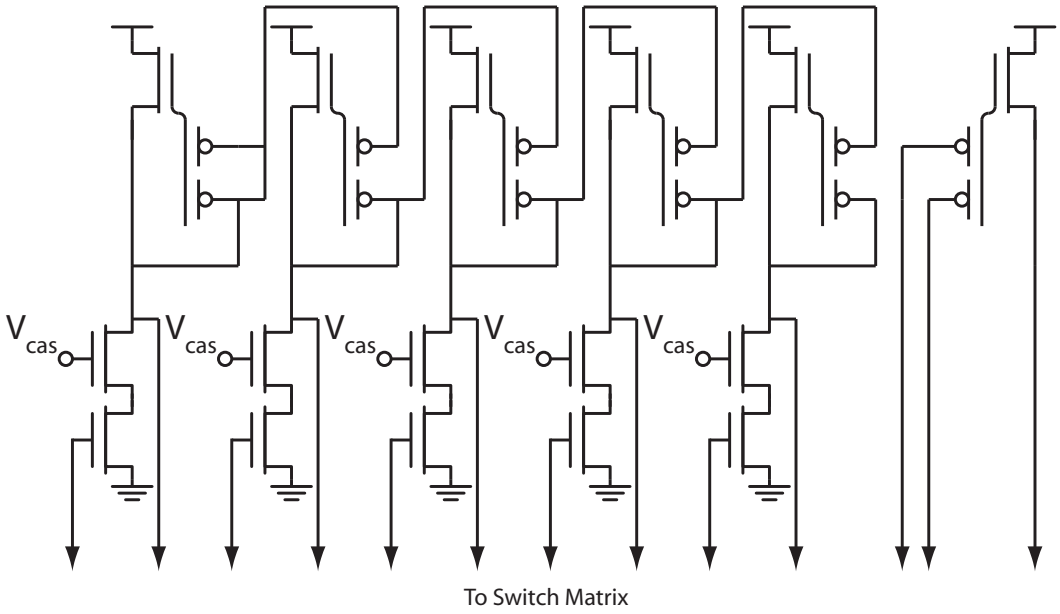


Figure 29. Basic MITE computation element of the MITE FPAA.

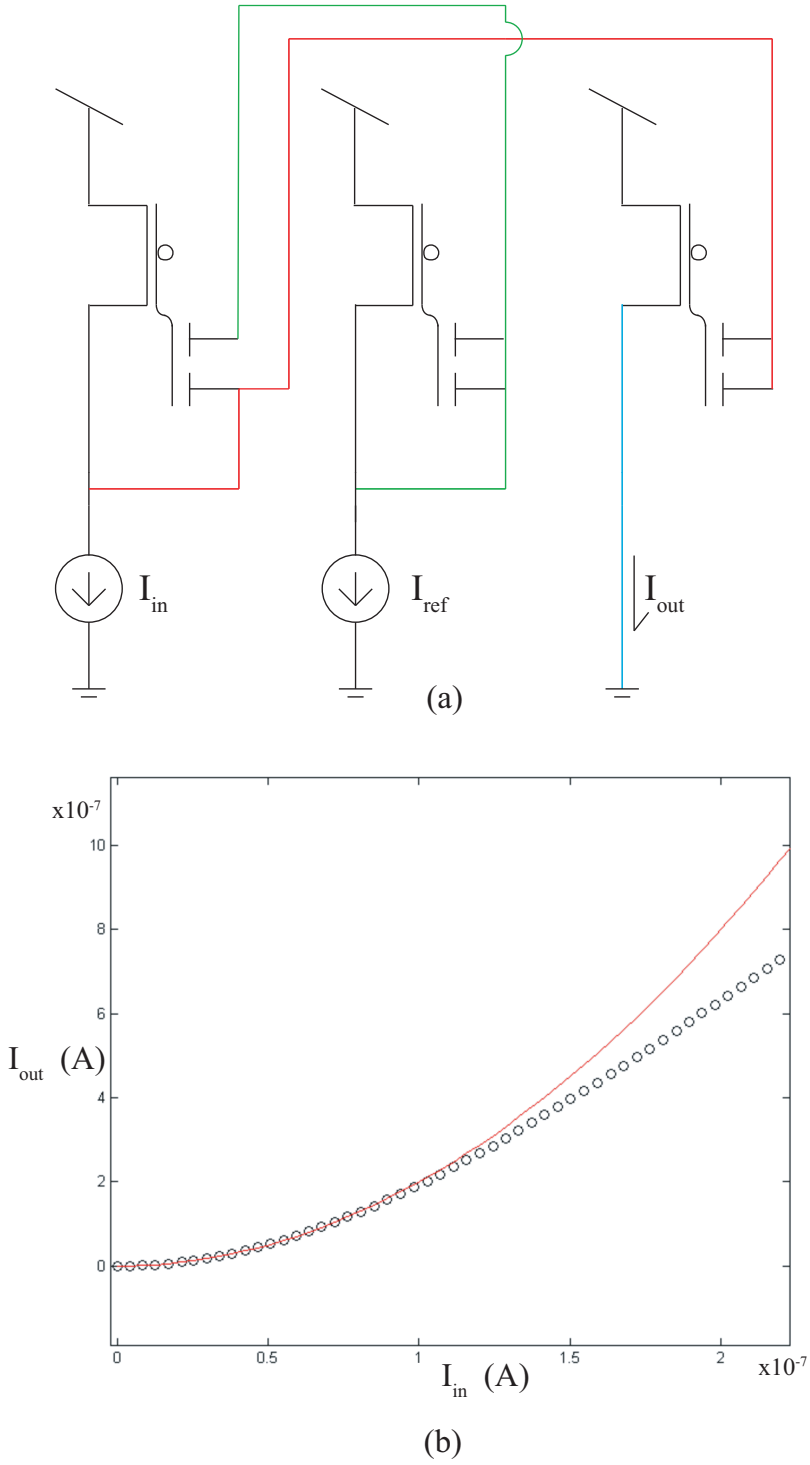


Figure 30. MITE FPAA squaring circuit. (a) Schematic of the three element translinear loop. (b) The bubbles are the data from the MITE FPAA and the red curve is an ideal square.

CHAPTER 5

TOOLS

In order to make RASP family of FPAAs more accessible, an entire chain of tools have been developed. By supporting higher-level design tools, a broader audience than just RASP specialists will be able to utilize this family of FPAAs. This will open up new opportunities to DSP and neuromorphic engineers who might not necessarily have the required expertise in circuit design to take advantage of an analog solution. One popular high-level signal processing design package in which many engineers feel comfortable is The Mathworks Matlab/Simulink, which we chose as the top-level medium for creating systems for the FPAA.

The design flow for implementing Simulink projects on the FPAA is shown in Figure 31. Analog systems can quickly and easily be made in Simulink by using the *sim2spice* compiler tool (shown in the dashed-box), which provides a custom Simulink library of computational blocks that have a corresponding analog implementation [26]. These custom blocks can then be connected together in a standard Simulink project, then simulated in Simulink to test higher level functionality, and converted to a Spice netlist.

Once a netlist is generated from *sim2spice*, the circuit can then be simulated in the Spice environment to test the device level functionality. With the functionality confirmed, the RASPER place and route tool can be used to create a targeting file for the FPAA. This targeting file is a list of switch addresses that can be viewed and modified with the RAT visualization tool and ultimately programmed onto the FPAA to create the hardware implementation of the system. In this Chapter, I describe each of the tools in greater detail as well as give examples of systems compiled down through the whole tool chain.

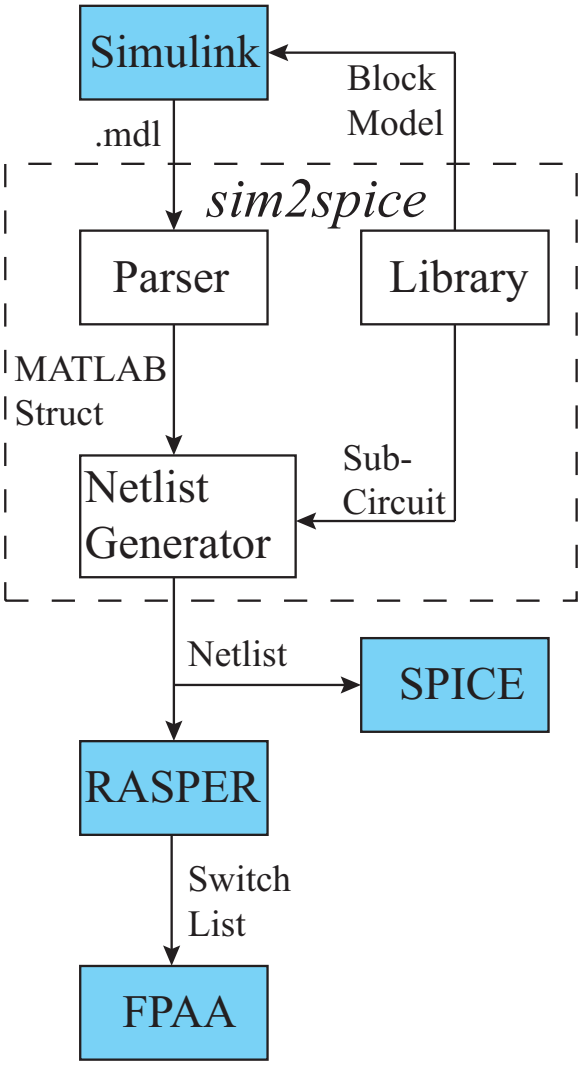


Figure 31. Sim2spice process flow.

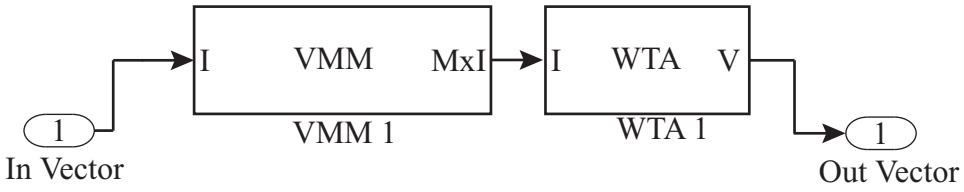


Figure 32. Example Simulink VMM and WTA. This system was designed using the custom library in sim2spice.

5.1 Simulink Tool

Simulink, a software product by The Mathworks, is a tool capable of modeling and simulating a large variety of signal processing systems based on an intuitive graphical block diagram system. It allows one to make high-level designs by simply dragging and dropping blocks into a diagram which are then easy to visualize, debug, and simulate. While there have been compilers developed for automating Simulink to digital FPGA circuit synthesis such as [27, 28], there was no existing tool that automatically compiles Simulink models to reconfigurable analog VLSI hardware, which I am presenting here.

Along with this automation tool, a library of custom analog signal processing blocks was also created. These blocks are able to run in the Simulink environment and contain a corresponding Spice circuit model. These blocks, such as the vector-matrix multiplier (VMM) and winner-take-all (WTA) in Figure 32, have specific analog circuit implementations that allow them to fully utilize the FPAA's strengths as discussed in Chapter 4.

The conversion process, shown in the dashed box of Figure 31, has three basic parts: the Parser, Netlist Generator, and the Component Library. The input to *sim2spice* is a Simulink model (.mdl) file, specifically from a design using the custom-written blocks, and the output is a Spice netlist. The netlist can then be used either by Spice to run simulations or compiled and targeted to the FPAA by RASPER.

5.1.1 Parser

The parsing is performed in Matlab with the use of a custom Python program. The input to the parser is the Simulink .mdl file, and the output is a Matlab structure containing the

```
Block {
  BlockType              Reference
  Name                "VMM 1"
  Ports               [1, 1]
  Position                  [390, 122, 500, 158]
  SourceBlock         "blocklib/VMM"
  SourceType          "VMM"
  size                "3"
  tau                 "0"
  elements                  "[0 2 3; 1 3 1; 5 5 5]"
}
Block {
  BlockType              Reference
  Name                "wta 1"
  Ports               [1, 1]
  Position                  [610, 338, 720, 372]
  SourceBlock         "blocklib/wta"
  SourceType          "WTA"
  size                "0"
}
```

Figure 33. Example .mdl file. This is a section of the .mdl file that results from the VMM/WTA system shown in Figure 32.

information needed to create a netlist. The Python program is packaged as an executable (.exe), which allows it to run without the installation of Python. The PyParsing module was used to to sort and make lists out the .mdl files [29]. Figure 33 shows a portion of of the .mdl file from the VMM/WTA block design, which the parser would interpret.

5.1.2 Netlist Generator

The netlist generator takes the Matlab structure created by parsing the Simulink .mdl file and converts it to a Spice netlist with the use of the component library. It makes several passes over the structure and keeps track of node numbering and connections between subcircuits representing parts of various Simulink blocks.

The netlist generator begins by reading an xml-type configuration file for the library listing the Simulink block types. Each block type has a corresponding xml-type configuration file listing parameter types and subcircuit definitions for that block. These parameter types include general parameters, such as block name and vectorized size, and parameters specific to certain blocks, such as the matrix elements for a VMM. Each block also has a

```
*spice file generated by sim2spice.

.INCLUDE fpaa_tech.sp

*INPORT
*>> pin io_lt 0 net int1net1p
*>> pin io_lt 1 net int1net1n

*INPORT
*>> pin io_lt 2 net int3net1

.SUBCKT vmmsub int1net1pInp int1net1nInn int3net1In int2net1pOutp int2net1nOutn
*VMM
XfgInp1 nRowp1 int1net1pInp FGE1 PARAMS: val=2e-007
xotap1 int3net1In int1net1pInp nRowp1 OTA PARAMS: Ib=0.001
XfgInn1 nRown1 int1net1nInn FGE1 PARAMS: val=2e-007
xotan1 int3net1In int1net1nInn nRown1 OTA PARAMS: Ib=0.001
XfgTL11 nRowp1 int2net1pOutp FGE1 PARAMS: val=2.4e-007
XfgTR11 nRowp1 int2net1nOutn FGE1 PARAMS: val=1.6e-007
XfgBL11 nRown1 int2net1pOutp FGE1 PARAMS: val=1.6e-007
XfgBR11 nRown1 int2net1nOutn FGE1 PARAMS: val=2.4e-007
.ENDS

Xvmmsub3 int1net1p int1net1n int3net1 int2net1p int2net1n vmmsub

*OUTPORT
*>> pin io_lt 3 net int2net1p
*>> pin io_lt 4 net int2net1n

*>> devicefile rasp2_8.dev
*>> project work
```

Figure 34. Example Spice code generated by sim2spice.

build script, which is a Matlab function specific to that block. It defines how the netlist is to be constructed given the parameters specified in its configuration file, the specific values for these parameters passed from Simulink through the netlist generator routine, and the Spice subcircuit primitives defined in its configuration file.

The netlist generator builds a data structure containing a list of the blocks and their connections, and then calls each block's custom build function, passing parameter values obtained from Simulink through the parser. Each custom build function uses the block's configuration file and the parameter values passed to it to write its netlist. Finally, the netlist generator writes the complete netlist to a Spice netlist file.

An example netlist generated by *sim2spice* is shown in Figure 34. This netlist is a simple 2×2 VMM which was compiled from Simulink. At the top, the .sp file that describes the CAB components is included, which allows the design to be simulated. Below that, the

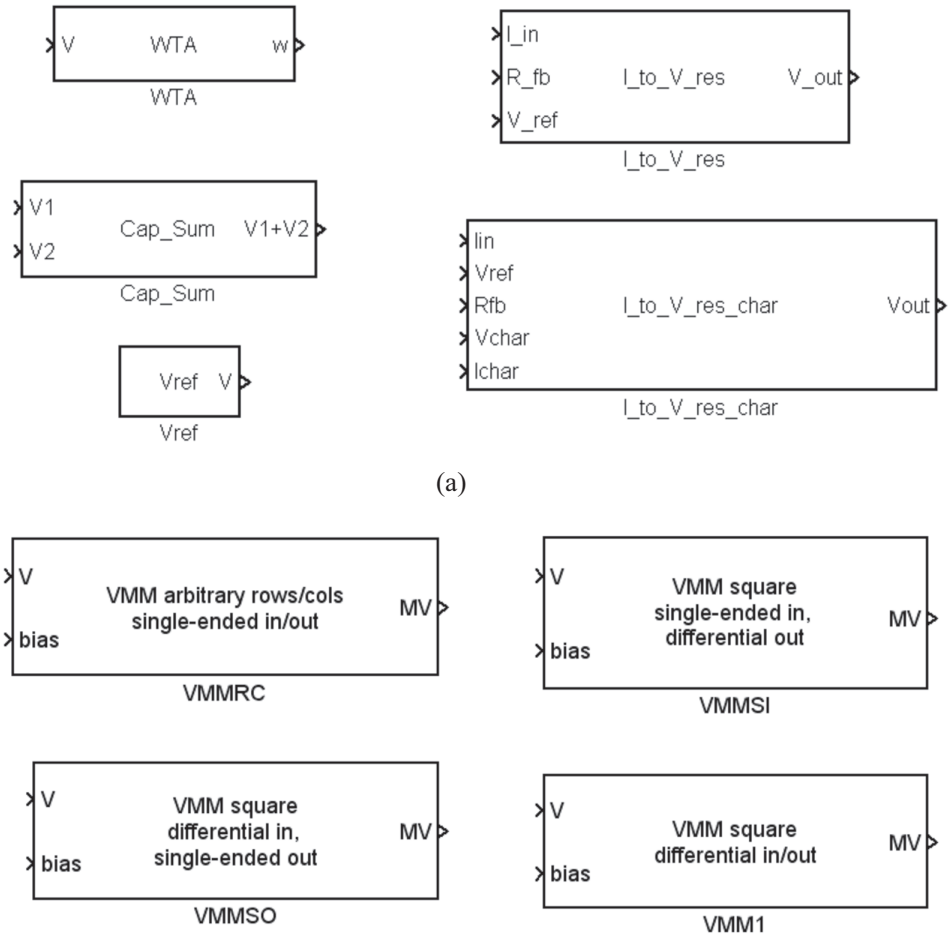


Figure 35. Simulink libraries. (a) Library of standard parts. (b) Library of VMMs.

inports are defined and mapped to I/O pins on the FPAA. Next, all of the subcircuits for each distinct block is generated (in this case, there is only one). Then, the subcircuits are instantiated with the correct net configuration and the outports are defined. Finally, it includes a device file, which RASPER uses to know which particular FPAA to place and route to.

5.1.3 Custom Library

The Simulink to FPAA library is composed of a set of Simulink blocks that simulate certain higher-level signal processing functionality and a corresponding set of circuit elements

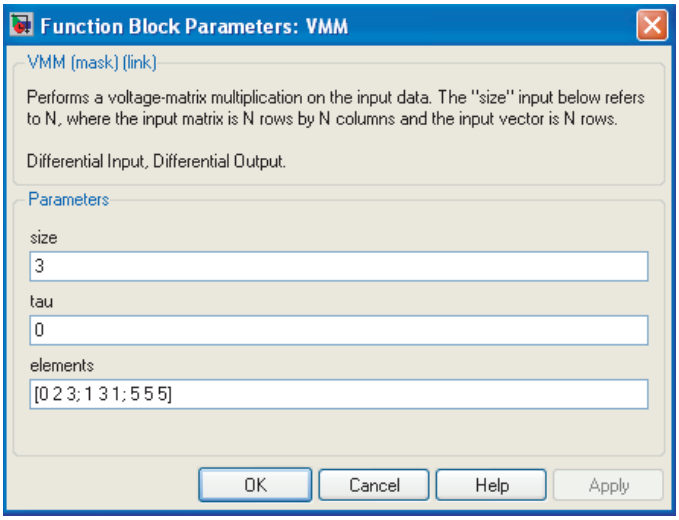


Figure 36. VMM properties box in sim2spice.

for Spice simulation. Since high-level Simulink block parameters can be varied, the corresponding Spice circuits are built up on-the-fly from basic circuit element sets for a specific Simulink block. Figure 35 shows a couple of the block libraries that are currently defined. The first is a library of standard parts containing many of the building blocks described in Chapter 4. The other library shown is solely for different varieties VMMs. In addition to these libraries, we have libraries for filters, hearing aid components, neuron channels, dendrites, and an AM demodulator components, among others.

Within Simulink, the behavior of a given custom block is defined by a Matlab S-function block with an associated Matlab m-file defining the block's time-domain behavior. The user, however, does not have to be concerned with the inner workings of a block and can simply place it in a Simulink model from the library, much like any built-in block. Each component calls a property box, shown in Figure 36, for easily editing relevant parameters such as dimension (size) and multiplier weights.

These libraries are meant to be built up over time to house a collection of functions that can be implemented as analog circuits. It will provide a standardized list of analog blocks and possible functions that can be implemented in analog as well as a platform for developing complex systems from basic analog primitives.

5.2 RASPER

RASPER, developed by Faik Baskaya, is the place and route tool used for targeting Spice netlists to the FPAA [30, 31, 32]. The output is a list of switch addresses and the values to which they should be programmed, given in the format: (row, column, prog value). The (row, col) address refers to the desired floating gate's location in the cross-bar matrix described in Chapter 3. The programmed value indicates if this floating gate is intended as a switch or a computational element. A programmed value above approximately $30\mu A$ will result in a switch that is all the way closed and any value below this will describe the amount of current that the transistor is programmed to pass with its source at V_{DD}. This list of switches can then be targeted directly onto the RASP family of FPAAs.

Figure 37 shows a 2×2 VMM (the schematic is in Figure 25) as implemented with switches in an FPAA CAB. Notice that this particular circuit demonstrates the use of switch elements for computation [13]. In the RASPER netlist input file, the particular FPAA is specified by the device (.dev) file. This .dev file describes all of the important attributes of a given FPAA such as: number and type of horizontal & vertical lines, CAB elements, and I/O lines. By including this file, RASPER will be compatible with future generations of FPAAs and routing structures.

5.2.1 Spice Library

Although the FPAA's CABs contain pre-defined circuits, in order for Spice to accurately simulate a design these CAB elements need to be implemented as subcircuits. Most of these are straight forward one-to-one mappings, such as the MOS elements, $500pF$, caps and T-gates. However, several CAB elements use floating gates as programmable current sources, in particular the OTAs. For these, an ideal current source was simply used in the subcircuit for simulation purposes and that value is then passed to the FPAA as the floating gate target value.

One problem that arose was how to model the floating-gate elements when they are explicitly used in the circuit. Examples of this are the floating-gate input OTAs, the MITEs,

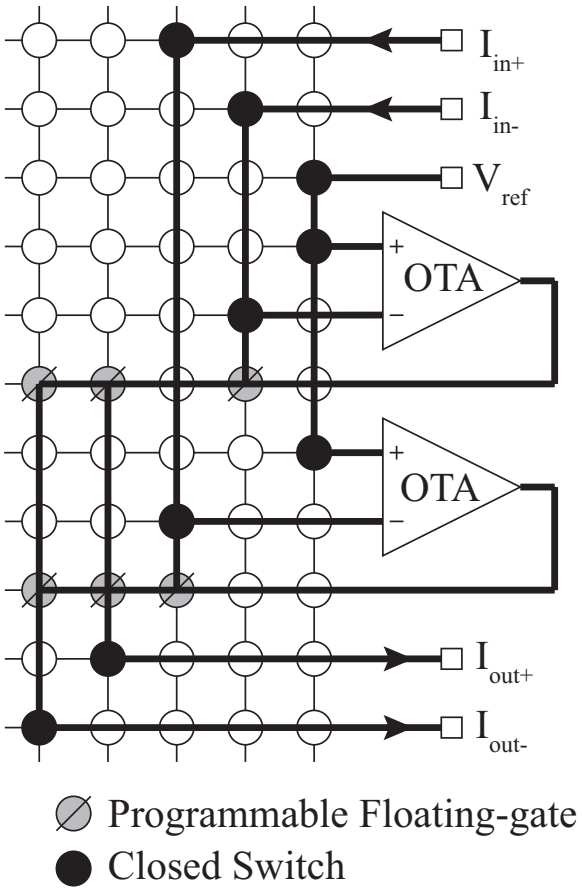


Figure 37. VMM implementation on FPAA. The use of floating-gate elements for computation is shown by the gray switches.

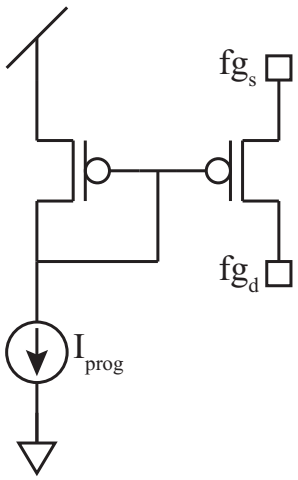


Figure 38. Floating gate Spice model.

and the switch elements used for computation. The problem arises from the rule that in Spice each node needs a DC path to ground, therefore gates cannot be left floating. There are several common models for floating-gate transistors [33, 34, 35]. However, in the interest of simplicity, other models were investigated.

One popular model is to simply place a DC voltage source on the gate through a large resistor ($10^{26}\Omega$). Although this achieves reasonable results, the DC voltage does not translate to the current which is actually programmed in an intuitive way. The model thus chosen was to force a DC current through an indirect transistor, as shown in Figure 38. This model was chosen because it allows for the programming current value to be directly used in Spice and the circuit closely resembles the actual schematic of the indirect system discussed in Chapter 2 [4]. In this implementation, the in-circuit FET's source and drain terminals are free to be tied to other nets but the gate is tied to the same potential as a pFET that is passing the programmed current with its source at V_{DD}.

5.2.2 RAT Visualization Tool

The Routing and Analysis Tool (RAT), developed by Scott Koziol and David Abramson, provides a graphical way to view the compiled circuits. This visualization tool has

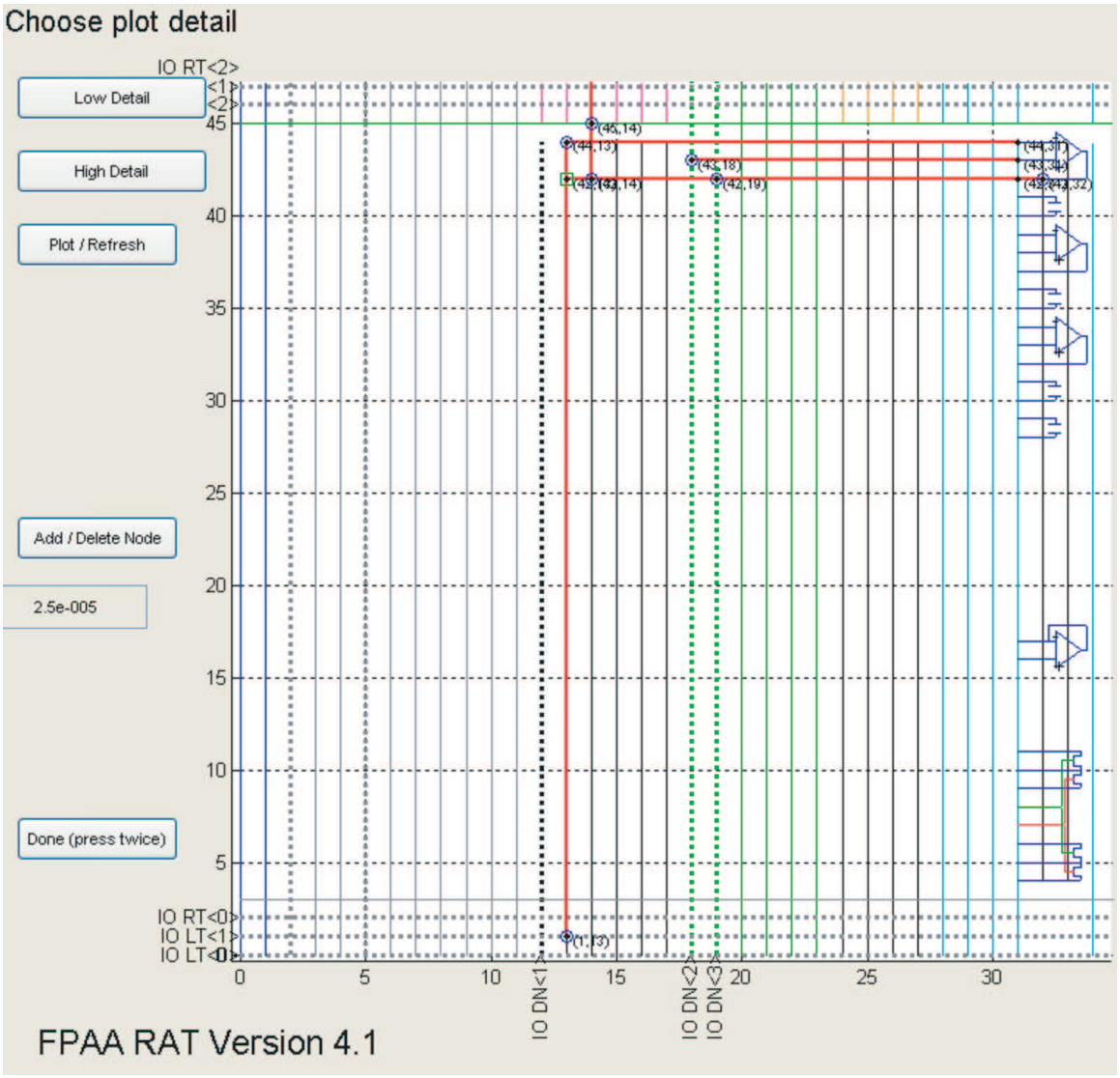


Figure 39. RAT design.

proved invaluable when designing and debugging on the FPAA because it has mostly eliminated the need to use fuse charts. The input to the RAT is a programming (.prg) file that includes the switch list in the form output by RASPER. By running the command $FPAA\_RAT\_main(filename.prg)$, the GUI of Figure 39 is launched. This window shows a zoomable image of the FPAA routing structure and CAB elements. The routing lines are color-coded by type and the I/O ports are clearly labeled. The switches from the input list appear as large black dots connecting the corresponding horizontal and vertical lines, and if a particular switch is used as a computational element, it is shown with a green circle around it. The lines connecting elements are highlighted in red to easily follow the connectivity of a particular net.

In addition to being able to view a circuit, modifications can be made to it. Switches can be added or deleted and the connectivity highlighting will be updated accordingly. Once modification is complete, the new design can be output into a new .prg file that has the same filename as the input file, but with "*_out*" appended to the end of it.

5.3 Evaluation Board

To program, communicate with, and test the RASP family of FPAAs the custom four-layer PCB in Figure 40 was built. This evaluation board communicates over and is fully powered by USB, but it has the capability to be powered by a 5V DC supply and communicate over a serial connection. The board is controlled by an ATMEL ARM microcontroller for handling instructions from the computer using Matlab commands. It also includes a 40-channel 14-bit DAC, a 4-channel 8-bit ADC, audio input/output amplifiers and jacks, and all of the programming circuitry not already on-chip. In order to have maximum control and flexibility, almost every signal is pinned out to a header, a map of which is given in Figure 41. All 52 FPAA I/O (4 to SMA connectors), the 40 DAC channels, 4 ADC channels and many of the microcontroller and programming lines are pinned out. The AV_{DD} plane is jumpered so power measurements can be taken.

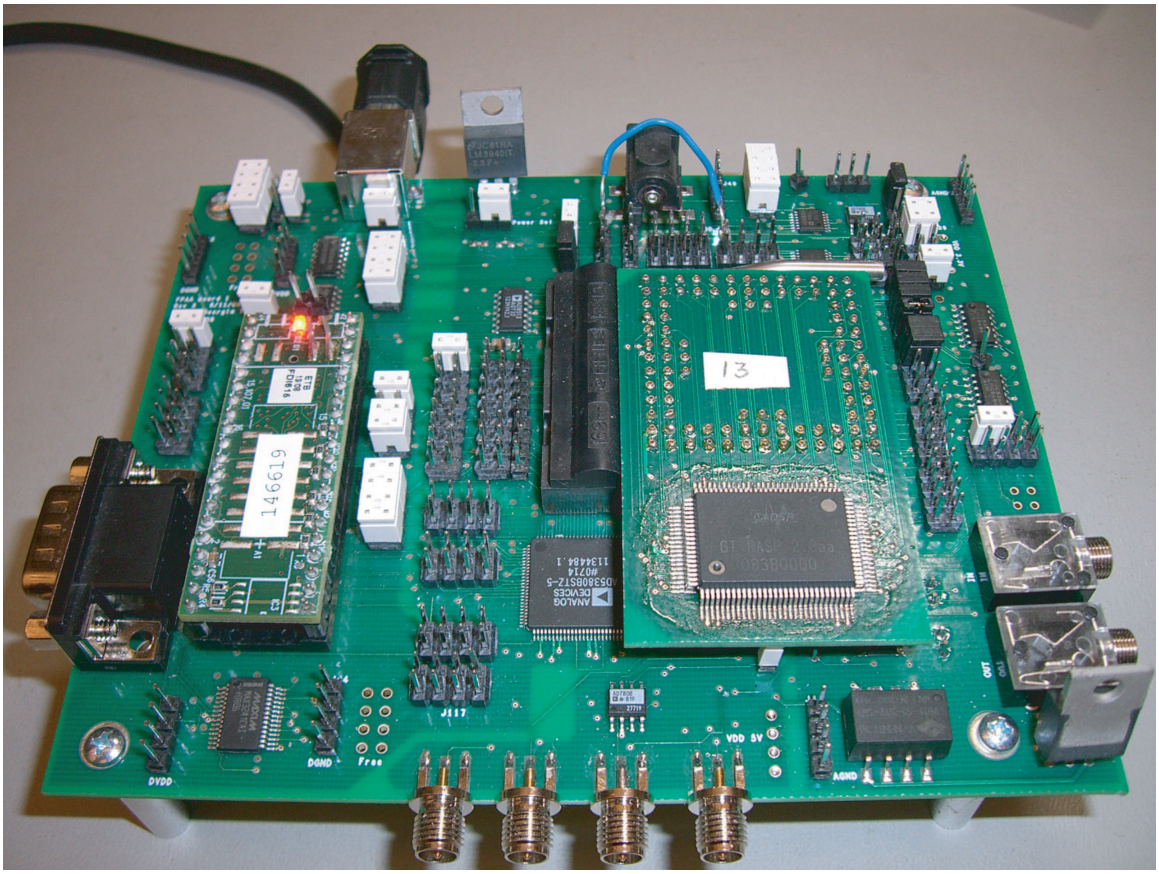


Figure 40. Evaluation board.

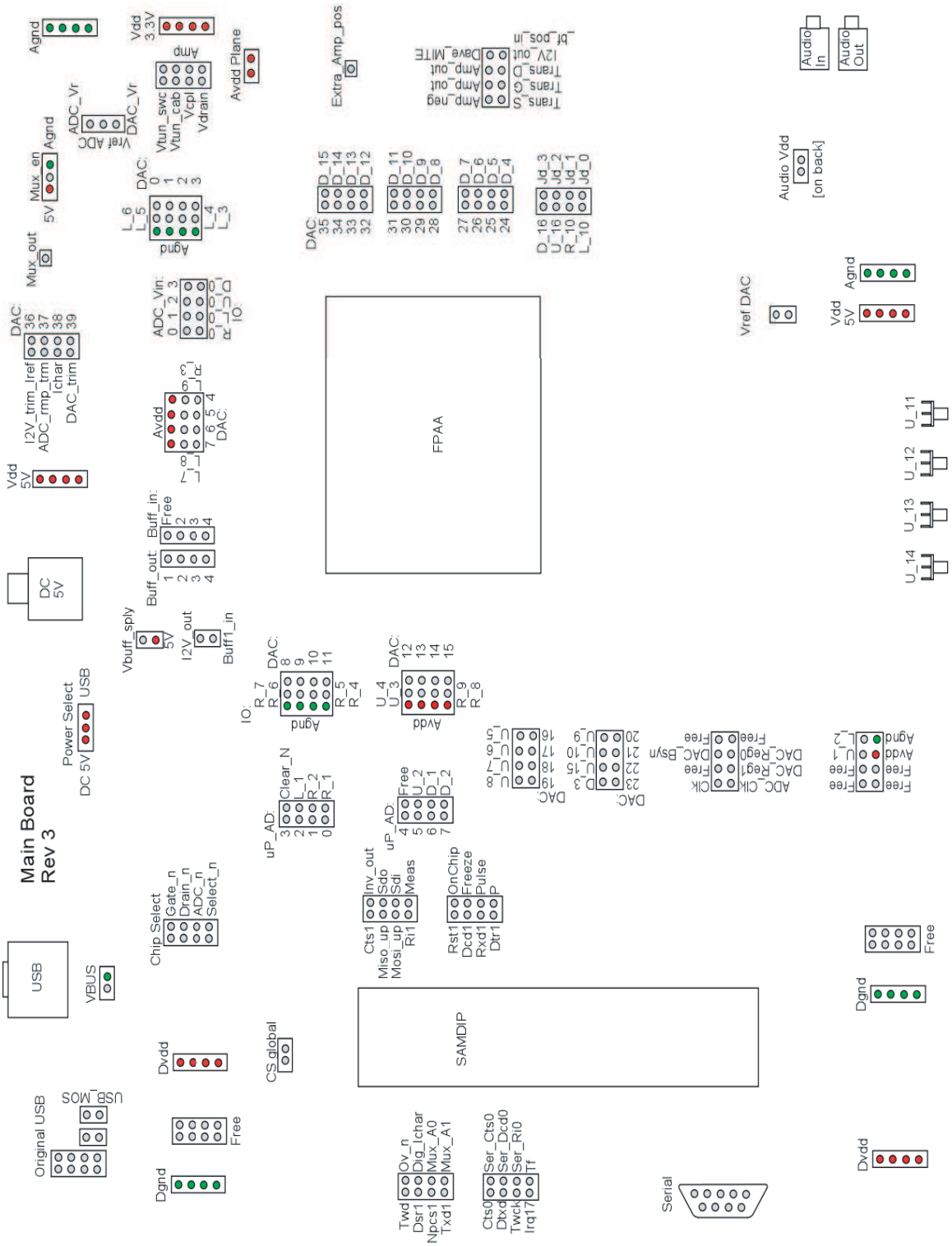


Figure 41. Header map.

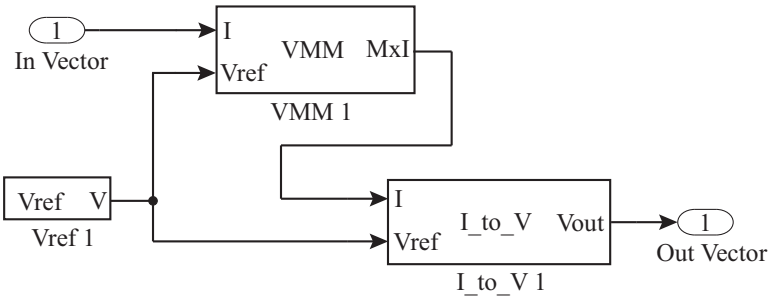


Figure 42. Simulink block level design. The VMM structure for the filter and DCT. The VMM, voltage source, and output stage are shown. The sim2spice tool was used to compile the netlist for the filter. This netlist was then targeted on the RASP 2.8a FPAA using the RASPER tool.

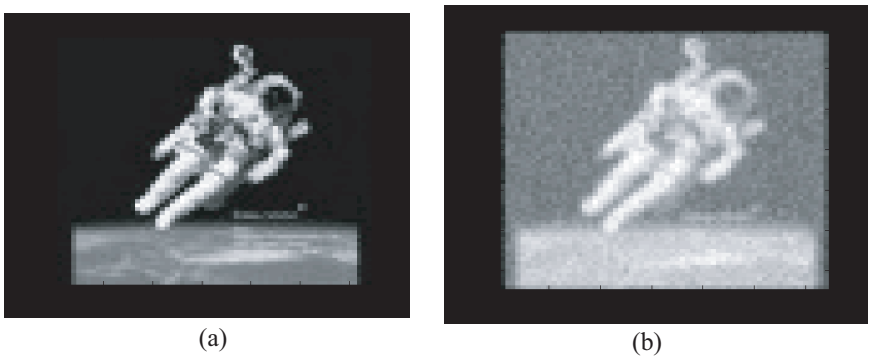


Figure 43. Gaussian convolution. (a) Original image. (b) Smoothed image by Gaussian smoothing filter on FPAA.

5.4 Example Systems

The Simulink test project for the example system, shown in Figure 42, includes a VMM, voltage source, and I to V blocks. Also in this diagram are in-ports and out-ports which correspond to I/O pins on the IC. Each of these examples were drawn at the Simulink level then compiled down to the FPAA.

5.4.1 2D Image Filtering with Gaussian Smoothing Filter

We programmed a 1×5 VMM on the FPAA with weights representing a Gaussian convolution filter of length 5, properly scaled to programmable values for floating-gate multipliers. We applied this filter to a 64×64 pixel test image, first to rows and then to columns of the input image, giving a 2D smoothing transform. Figure 43 shows the input image and the output image after having been passed through the Gaussian smoothing filter on the FPAA.

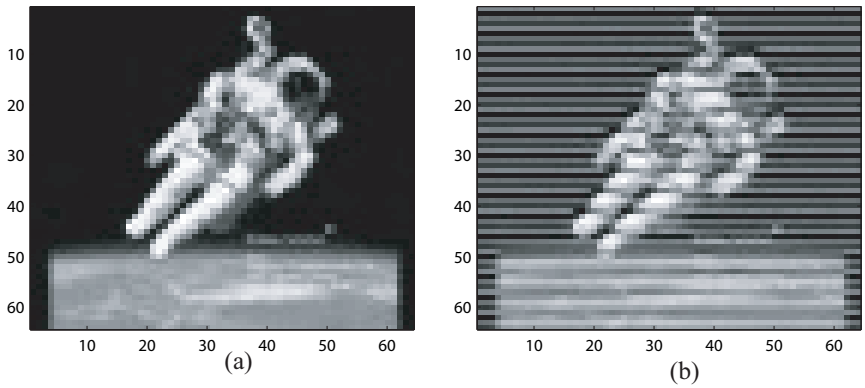


Figure 44. Discrete cosine transform. (a) Original image. (b) DCT on FPAA and inverse in Matlab.

5.4.2 Discrete Cosine Transform

We programmed a VMM with weights corresponding to a 4×4 discrete cosine transform (DCT) matrix. It was necessary to shift and scale the weights in order for them to be in the right range (0.5 to 1.5) for programming on the FPAA, an operation which can be reversed once the transform is applied in order to get the actual DCT transform of the output. We applied the resulting 4×4 DCT VMM as a block transform horizontally across each row of height 4 in a grayscale 64×64 pixel sample image. We then took the inverse transform with Matlab. The input image and resulting output image are shown in Figure 44. The DCT transform worked as desired on the FPAA hardware, with some level of noise resulting from imprecise input current measurements.

CHAPTER 6

CONCLUSION

In this Thesis, I addressed the importance of analog signal processing and how FPAAs can enable its use. In Chapter 2, I reviewed our fundamental piece of technology, the floating-gate transistor. I defined its device characteristics and the ways in which we use tunneling and injection to program it. I also illustrated how we can target a specific transistor for injection by arranging the devices in 2-D arrays.

In Chapter 3, I reviewed the current state of FPAAs and gave a detailed description of the RASP family. I described the main advantages of FPAAs and explained how important floating gates are to creating a large-scale reconfigurable device. I also elaborated on three of the more recent RASP FPAAs: the general-purpose 2.8a and 2.9a, and the high-speed 2.9b.

In Chapter 4, I detailed several important analog building blocks and their FPAA implementation. These building blocks form a base for analog signal processing on the FPAA. I began with two basic computation blocks: the winner-take-all and the capacitive summer. I then went into the interfacing blocks: the V-to-I and I-to-V. I discussed the implementation of constant current and voltage sources using common FPAA components. I then explained how vector-matrix multiplication is a function well suited to the FPAA, and detailed its implementation. Lastly, I illustrated the capabilities of MITEs on both the general-purpose FPAA and the dedicated MITE FPAA.

Finally, in Chapter 5, I outlined an entire chain of tools which can be used to create FPAA designs. At the top level, *sim2spice* was used to compile Simulink designs into a Spice netlist. Next, I detailed how this netlist can be run through RASPER to get the FPAA targeting code which can then be visualized with the RAT or programmed onto the hardware. This Chapter concluded with two example systems which were compiled through the complete tool chain.

6.1 Personal Contributions

Much of the work in this Thesis is my own. However, almost everything was done in a collaborative effort. For my contribution, I helped with the early testing of the RASP 2.8a along with the rest of the ICE lab: Arindam Basu, Stephen Brink, Scott Koziol, Csaba Petre, and Shubha Ramakrishnan. I designed and tested many circuits with several versions of programming code, boards, and two ICs (the 2.8a and the post-mask-change 2.8aa). We also held several FPAA workshops where we taught others how to program and use the FPAA. For the RASP 2.9a, I had a large part in the porting of the 2.8a components doing schematic simulations and layout. For 2.9b, I completely designed and laid it out as a modification of 2.9a. I synthesized, programmed onto the FPAA, tested, and created the plots for all of the analog building blocks discussed in Chapter 4. The *sim2spice* tool was created in collaboration with Csaba Petre. I wrote the parsing script used to read the .mdl files and extensively developed the library, while Csaba wrote and has been maintaining the netlist generator, as well as creating library blocks. The evaluation board was designed with input from the whole ICE lab, with Scott and myself doing the schematic capture and layout. The board has been through several revisions and a couple spin-offs have been tried.

6.2 Future Work

As demonstrated in this Thesis, a tremendous amount of work has already been completed in creating a reliable reconfigurable system. However, there are still several areas where work is left to do. First, the *sim2spice* library development is an open-ended task. As each new block is developed, it goes into a library which is shared by everybody. The hope is that as more people trust and are comfortable with this system, they will include making a Simulink block as a standard part of their design. The high-speed FPAA's PCB still needs to be built and the IC tested. The IC has already been fabricated, and the board has been designed on paper, but it still needs to be built and the tests still need to be run. Also, there is testing to be done on the MITE FPAA. Several static circuits have been compiled, but the

dynamics have not been as thoroughly demonstrated. Opportunities here lie in designing bandpass filters, oscillators, and ODE solvers. We are in the process of getting the new MITE FPAA packaged, which underwent a mask change. This should help with future testing. In addition to testing the MITE FPAA, there are some things that make sense to incorporate into a new version of the IC. The modifications would be to add a current splitter into the CABs because they are frequently used and take up a lot of resources if they need to be compiled, and designing a new MITE altogether. Since Dave Abramson documented a strong kappa variation with his gate-driven subthreshold pFET, it would be worthwhile investigating the practicality of incorporating Kofi Odame's source-driven devices into the CABs [36].

REFERENCES

[1] G. Frantz, "Digital signal processor trends," *IEEE Micro*, vol. 20, pp. 52 – 59, 2000.

[2] D. Kahng and S. Sze, "A floating-gate and its application to memory devices," *The Bell System Technical Journal*, vol. 46, no. 4, pp. 1288–1295, 1967.

[3] P. D. Smith, M. Kucic, and P. Hasler, "Accurate programming of analog floating-gate arrays," in *IEEE International Symposium on Circuits and Systems*, vol. 5, p. 489492, May 2002.

[4] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler, "Indirect programming of floating-gate transistors," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 951 – 963, May 2007.

[5] E. Lee and P. Gulak, "A cmos field-programmable analog array," *Solid-State Circuits, IEEE Journal of*, vol. 26, pp. 1860 – 1867, Dec. 1991.

[6] S. Koneru, E. Lee, and C. Chu, "A flexible 2-d switched-capacitor fpaa architecture and its mapping algorithm," in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, pp. 296 – 299, Aug. 1999.

[7] H. Kutuk and S.-M. Kang, "A field-programmable analog array (fpaa) using switched-capacitor techniques," in *Circuits and Systems, 1996. ISCAS '96., 'Connecting the World'., 1996 IEEE International Symposium on*, pp. 41 – 44, May 1996.

[8] Anadigm, "Dynamically reconfigurable fpaa with basic i/o," tech. rep., http://www.anadigm.com/ doc/DS030100-U008.pdf, Nov 2005.

[9] M. Fakhfakh, S. Masmoudi, and M. Loulou, "Towards a switched current fpaa," in *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, pp. 3 – 6, Dec. 2007.

[10] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli, "A field-programmable analog array of 55 digitally tunable otas in a hexagonal lattice," *Solid-State Circuits, IEEE Journal of*, vol. 43, pp. 2759 – 2768, Dec. 2008.

[11] C. Looby and C. Lyden, "Op-amp based cmos field-programmable analogue array," in *Circuits, Devices and Systems, IEE Proceedings*, vol. 147, pp. 93 – 99, April 2000.

[12] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Developing large-scale field-programmable analog arrays," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 142, April 2004.

[13] C. Twigg, J. Gray, and P. Hasler, "Programmable floating gate fpaa switches are not dead weight," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 169 – 172, May 2007.

[14] T. S. Hall, *Field-Programmable Analog Arrays: A Floating-Gate Approach*. PhD thesis, Georgia Institute of Technology, July 2004.

[15] C. M. Twigg, *Floating Gate Based Large-Scale Field-Programmable Analog Arrays for Analog Signal Processing*. PhD thesis, Georgia Institute of Technology, July 2006.

[16] D. Abramson, *A mite based translinear fpaa and its practical implementation*. PhD thesis, Georgia Institute of Technology, Nov. 2008.

[17] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "Rasp 2.8: A new generation of floating-gate based field programmable analog array," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pp. 213 – 216, Sept. 2008.

[18] A. Basu and P. Hasler, "A fully integrated architecture for fast programming of floating gates," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 957 – 960, May 2007.

[19] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of o(n) complexity," *Adv. Neural Inf. Process. Syst*, vol. 1, p. 703711, 1989.

[20] C. Alexander and M. Sadiku, *Fundamentals of Electronic Circuits*. McGraw-Hill, 2 ed., 2004.

[21] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler, "A 531 nw/mhz, 128x32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity," in *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pp. 651 – 654, Oct. 2004.

[22] A. Bandyopadhyay, J. Lee, R. Robucci, and P. Hasler, "Matia: a programmable 80 uw/frame cmos block matrix transform imager architecture," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 3, pp. 663 – 672, 2006.

[23] R. Robucci, J. Gray, D. Abramson, and P. Hasler, "A 256x256 separable transform cmos imager," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 1420 – 1423, May 2008.

[24] B. A. Minch, "Synthesis of static and dynamic multiple-input translinear element networks," Tech. Rep. CSL-TR-2002-1024, Cornell University, June 2002.

[25] S. Subramanian, D. Anderson, P. Hasler, and B. Minch, "Optimal synthesis of mite translinear loops," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 2822 – 2825, May 2007.

[26] C. Petre, C. Schlottmann, and P. Hasler, "Automated conversion of simulink designs to analog hardware on an fpaa," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 500 – 503, May 2008.

[27] B. Sbarcea and D. Nicula, "Automatic conversion of matlab/simulink models to hdl models," in *International Conference on Optimization of Electrical and Electronic Equipment*, 2004.

[28] C. Chang, J. Wawrzynek, and B. Brodersen, "From bee to bee2: Development of supercomputer-in-a-box," in *presentation from Berkeley Wireless Research Center, University of California, Berkeley*, Dec. 2 2004.

[29] P. McGuire, "Pyparsing." http://pyparsing.wikispaces.com/.

[30] F. Baskaya, S. Reddy, S. K. Lim, and D. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," in *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, pp. 906 – 910, Aug. 2006.

[31] F. Baskaya, B. Gestner, C. Twigg, S. K. Lim, D. Anderson, and P. Hasler, "Rapid prototyping of large-scale analog circuits with field programmable analog array," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pp. 319 – 320, April 2007.

[32] I. Baskaya, S. Reddy, S. K. Lim, and D. Anderson, "Hierarchical placement for large-scale fpaa," in *Field Programmable Logic and Applications, 2005. International Conference on*, pp. 421 – 426, Aug. 2005.

[33] J. Gray, R. Robucci, and P. Hasler, "The design and simulation model of an analog floating-gate computational element for use in large-scale analog reconfigurable systems," in *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, pp. 253 – 256, Aug. 2008.

[34] T. Ochiai and H. Hatano, "Dc characteristic simulation for floating gate neuron mos circuits," in *Electronics Letters*, vol. 35, pp. 1505 – 1507, Sept 1999.

[35] K. Rahimi, C. Diorio, C. Hernandez, and M. Brockhausen, "A simulation model for floating-gate mos synapse transistors," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, pp. 532 –535, May 2002.

[36] K. Odame, E. McDonald, and B. Minch, "Highly linear, wide-dynamic-range multiple-input translinear element networks," in *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 2, pp. 2036 – 2040, Nov. 2003.